

# Detección automática de personas mediante Histograma de Gradientes Orientados



Grado en  
Ingeniería Informática

Trabajo Fin de Grado

Daniel Catalán Vitas

Daniel Paternain Dallo

Pamplona, 22/06/2017

## Índice

<b>1. Introducción</b>	1
<b>2. Base Teórica</b>	4
2.1 Detección de objetos	4
2.2 Extracción de características	5
2.2.1 Características basadas en intensidad	6
2.2.2 Características basadas en la textura	7
2.2.3 Características basadas en la forma	7
2.2.4 Histograma de Gradientes (HOG)	8
2.2.5 Local Binary Pattern (LBP)	12
2.3 Aprendizaje máquina y clasificadores	14
2.3.1 Máquinas de Soporte Vectorial (SVM)	19
2.3.2 Regresión lineal y logística	20
2.3.3 K-Nearest Neighbor KNN (Vecinos más cercanos)	22
2.3.4 Perceptrón Multicapa (MLP)	25
<b>3. Implementación</b>	30
3.1 Dataset de imágenes	30
3.2 Estructura general del algoritmo	32
3.2.1 Fase de entrenamiento	32
3.2.2 Fase de ejecución y test	34
3.3 Desarrollo general del algoritmo	37
3.3.1 Entorno de desarrollo	37
3.3.2 Estructura del código	38
<b>4. Resultados</b>	43
4.1 Métricas y parámetros	43
4.2 Resultados del clasificador base	44
4.3 Umbral de detección	46
4.4 Factor de escala	48
4.5 Pre-procesamiento y extracción de características	50
4.5.1 Escala de grises y color	50
4.5.2 Corrección de color	51
4.5.3 Computación del HOG	52
4.5.4 Reemplazar HOG por LBP	53
4.2 Clasificadores y sus parámetros	54
4.2.1 SVM: Parámetro C	54
4.2.2 Regresión Logística	56

4.2.3	Perceptrón Multicapa.....	57
4.2.4	KNN. Vecinos más cercanos. ....	58
<b>5.</b>	<b>Aplicación Práctica.....</b>	<b>60</b>
5.1	Solo el detector .....	60
5.2	Detector + tracking.....	63
<b>6.</b>	<b>Conclusiones .....</b>	<b>66</b>
6.1	Conclusiones técnicas .....	66
6.2	Trabajo futuro .....	67
<b>7.</b>	<b>Anexo .....</b>	<b>68</b>
7.1	Lista de figuras.....	68
7.2	Lista de tablas.....	70
<b>8.</b>	<b>Referencias .....</b>	<b>71</b>

## 1. Introducción

Los ordenadores se han convertido en algo indispensable en nuestro día a día. Son muy útiles en gran cantidad de áreas de trabajo e investigación porque pueden realizar tareas específicas complejas con mayor precisión y eficiencia que cualquier humano. Sin embargo, todavía no son capaces de realizar eficientemente algunas tareas que requieren de un mayor nivel de inteligencia, como puede ser el reconocimiento vocal, el razonamiento e interpretación lógica, o el análisis visual que los humanos realizamos de forma subconsciente y natural muchas veces a lo largo del día.

El desarrollo de sistemas de visión artificial que realizan tareas de reconocimiento visual es una de las áreas de investigación más activas desde hace unos años. A simple vista, un humano puede localizar un objeto sin esfuerzo, y de forma instantánea. Además, esta habilidad no se limita a reconocer que se está visualizando un objeto, sino que también somos capaces de clasificarlo, ya sea un coche o una persona, independientemente de su variación de color, pose, textura, deformación, oclusión, iluminación y complejidad del fondo. Para una máquina, esta es una tarea para nada trivial.

Gran parte de los investigadores que trabajan en visión artificial están centrados en desarrollar sistemas de reconocimiento que sean capaces de determinar con exactitud la identidad del objeto observado. Una de las principales tareas para alcanzar tal objetivo es la implementación de un detector preciso, lo que implica localizar, en términos de posición y escala, un objeto específico en una imagen estática.

La detección de personas es un caso particular de detección de objetos donde la clase objetivo son personas. Su interés radica en el potencial impacto positivo en la calidad de las aplicaciones que se beneficiarían de dicha tecnología (sistemas de vigilancia y seguimiento, seguridad en la automoción, robótica, análisis de contenido multimedia, interfaces de interacción avanzadas, entre otros).

Aunque ya se han propuesto muchas soluciones a este problema, la detección de personas sigue siendo una tarea complicada debido a la alta variedad de apariencias, poses y condiciones contextuales que puede adoptar una persona. Es más, también es un área competitiva dado que la alta investigación en este campo eleva continuamente los límites de eficiencia y precisión. Los métodos de detección de personas más conocidos hacen uso de una combinación de extracción de características y entrenamiento de clasificadores (SVM, Boosting, Random forest, Regresión logística...)

Aunque últimamente se han realizado muchos avances importantes, la detección de personas sigue ofreciendo un amplio rango de mejora, principalmente en términos de precisión y eficiencia.

Por estas razones, el objetivo de este TFG es desarrollar y evaluar un detector de personas. En particular, el trabajo tiene tres objetivos diferentes:

- Construir un detector desde cero usando distintos algoritmos, métodos, características y clasificadores, para después evaluar su eficiencia, tanto en términos de velocidad como de precisión. Un detector de personas se puede entender como la combinación de dos bloques principales, un algoritmo de extracción de características y un método de clasificación que usa esas características para decidir entre objeto / no objeto. En nuestro detector usaremos Histogramas de Gradientes (HOG) y Local Binary Patterns (LBP) como características, y Máquinas de soporte Vectorial (SVM), perceptrón multicapa (MLP), regresión logística y Vecinos más cercanos (KNN) como clasificadores.
- Realizar un análisis exhaustivo de cómo los distintos parámetros de entrenamiento, clasificadores, y características afectan al proceso de aprendizaje y al rendimiento final del detector. En el análisis se tendrá en cuenta las variables más significativas, como pueden ser el número de muestras de entrenamiento, los parámetros del extractor de características (HOG), los parámetros del clasificador (SVM), o la técnica de selección de características. Se decidirá qué algoritmo se comporta mejor comparando test estadísticos.
- Utilizar el detector con el clasificador y los parámetros que mejor resultado han obtenido en un caso práctico, como pueden ser fragmentos de video de una cámara de seguridad o de otra índole para analizar la versatilidad de detector y la viabilidad de la idea en su conjunto. También se probará a combinar las técnicas de detección con otras ideas que faciliten el trabajo, como puede ser el análisis de los cambios entre frames consecutivos del vídeo para detectar movimiento, o la reducción espacial y dimensional de imágenes para ahorrar tiempo de computación.

El resto de la memoria está organizado de la siguiente forma:

- **Apartado 2. Base teórica.** Primero introducimos algunas nociones generales sobre la clasificación de imágenes; luego explicamos que es la extracción de características, algunas de las más importantes y las que vamos a utilizar; después también explicamos que es el aprendizaje formal, y que algoritmos o clasificadores son más remarcables en la actualidad, además de detallar los más relevantes para nuestra tarea.
- **Apartado 3. Implementación.** En este apartado exponemos el trabajo realizado con el detector. Primero, describimos el diseño general del detector y después ofrecemos detalles de la implementación y de la estructura del código.
- **Apartado 4. Resultados.** Exponemos los resultados obtenidos con el detector, explicando todas las configuraciones evaluadas para los distintos algoritmos, además de analizar los resultados más importantes.
- **Apartado 5. Aplicación práctica.** Probamos los algoritmos y parámetros que han ofrecido un mejor resultado con casos prácticos, como puede ser la secuencia de vídeo de una cámara de seguridad.
- **Apartado 6. Conclusiones.** Resumen de los resultados y análisis estadístico de los mismos. También se exponen las ideas y direcciones a tomar en base a futuras líneas de trabajo.

## 2. Base Teórica

### 2.1 Detección de objetos

La **detección de objetos** hace referencia a un problema computacional cuyo objetivo es detectar y localizar (buscar) objetos de cierto tipo (clase) en imágenes estáticas o en frames de videos [1].

Es importante clarificar, antes de avanzar más, los siguientes conceptos básicos del procesamiento de imágenes relacionado con la detección de objetos (además asumiremos que la clase de objeto que nos interesan son las personas).

- Detección de la presencia de objetos: determinar si en la imagen aparecen una o más instancias del objeto que se está buscando, sea cual sea su escala y posición. ¿Hay personas en la imagen?
- Detección / localización de objetos: Determinar la localización y escala específicas de una o más instancias del objeto que se está buscando. ¿Dónde aparecen las personas en la imagen, si es que las hay?
- Reconocimiento de objetos: Determinar la identidad de cada instancia particular del objeto detectada en la imagen. ¿Es Kate la persona de la imagen?
- Categorización de objetos: Determinar a qué clase pertenece cada instancia detectada en la imagen. ¿Es esto una persona, una mesa, o un coche?

Los sistemas de detección de objetos se componen habitualmente de dos fases diferenciadas, que en este trabajo analizamos por separado: la extracción de características y la clasificación. En la siguiente figura se puede apreciar una vista general de la estructura del sistema.



Figura 1. Esquema general de un sistema de detección de imágenes.

En el presente trabajo el cuerpo humano es la clase de interés. El objetivo es encontrar todas las instancias de seres humanos presentes en una imagen. Esta tecnología tiene aplicaciones en numerosas áreas de la visión artificial.

## 2.2 Extracción de características

Una **característica** es un atributo o propiedad individual de un objeto que es relevante a la hora de describirlo y reconocerlo, y que permite distinguirlo de otros objetos.

La **extracción de características** es el proceso de transformación de los datos de entrada (usualmente muchos y redundantes) en una representación reducida, conocida como conjunto de características [2]. Es decir, es una técnica de reducción de la dimensionalidad. El proceso extrae un subconjunto de nuevas características a partir del conjunto original mediante alguna función manteniendo la mayor cantidad de información relevante posible. Un vector de características es un conjunto de características agrupadas.

La extracción de características es clave en muchos algoritmos de aprendizaje formal dado que condiciona directamente su rendimiento, y se usa como punto de partida en gran cantidad de algoritmos de visión artificial. No solo se tiene en cuenta la relevancia de los datos, también otros aspectos como el volumen de los mismos. El tamaño del vector de características condiciona la eficiencia, velocidad, complejidad y precisión del algoritmo. Por lo tanto, la efectividad y eficiencia de la extracción y selección de características es un paso crucial en el análisis de imágenes y, especialmente, en el reconocimiento y la detección de objetos. En este caso particular, los datos de entrada son una imagen y las características son descriptores visuales principalmente basadas en color, textura o forma.

De acuerdo a su complejidad, las características de las imágenes se pueden clasificar en dos grupos. Las de *bajo nivel* son características básicas que se pueden extraer directamente a partir de la información de los píxeles de la imagen. Las de *alto nivel* requieren un pre-procesamiento y están habitualmente basadas en otras de bajo nivel. También se las puede clasificar de acuerdo a la dimensión original, o procedencia de los datos extraídos. Las características globales son aquellas que proceden de la imagen completa, y las características locales son las que representan solo a una pequeña porción de la imagen. Las características globales son mucho más sensibles que las locales a los cambios en la apariencia del objeto (pose, oclusión y condiciones de iluminación)



### 2.2.1 Características basadas en intensidad

Este tipo de características están basadas directamente en el valor absoluto de los píxeles tanto de imágenes a color como en escala de grises. La intensidad es una característica de bajo nivel, y además es invariante a la escala, algo esencial para la caracterización de las imágenes. Su uso está muy extendido dada su relevancia y simplicidad. Sin embargo, la intensidad sola no provee una descripción completa de la imagen, sobre todo porque no posee información estructural. Además, por lo general, no son las características más eficientes computacionalmente hablando, ya que son atributos de los píxeles, y varían con la dimensión de la imagen.

La **comparación de la intensidad por pixel** es la más simple característica basada en intensidad y consiste en comparar la intensidad de dos píxeles [3]. La salida es un único número por comparación. Se pueden comparar todos los pares de píxeles en una imagen, pero por lo general, para reducir la dimensión del vector, solo se comparan unos pocos.

$$f = \begin{cases} 1, & I(x_i, y_i) < I(x_j, y_j) \\ 0, & \text{en otro caso} \end{cases}$$

La **diferencia en la intensidad del pixel** es otra característica derivada que consiste en computar la diferencia de intensidad entre dos píxeles. En este caso, la salida puede ser la propia diferencia o también un valor binario asignado de acuerdo a cierto umbral.

$$f = I(x_i, y_i) - I(x_j, y_j)$$

$$f(x) = \begin{cases} 1, & I(x_i, y_i) - I(x_j, y_j) > \text{umbral} \\ 0, & \text{en otro caso} \end{cases}$$

La **intensidad media** es otra característica que simplemente mide la intensidad media en una región específica de la imagen.

Estas características también se pueden clasificar dependiendo de si la imagen es en color o en blanco y negro. Las características basadas en el color se definen de acuerdo a un modelo de color particular, como por ejemplo RGB, LUV, CIE Lab o HSV. Las características más importantes de este tipo que se han propuesto hasta ahora son: *color histogram*, *color moments* (CM), *color coherence vector* (CVV) y *color correlogram*.

Las características basadas en escalas de gris provienen de imágenes en blanco y negro, donde el valor de cada pixel es un único valor entre 0 y 255 que representa la intensidad de gris. La característica más destacada de este tipo es la Transformada de Haar, una secuencia de funciones con una estructura matemática para describir patrones.

### 2.2.2 Características basadas en la textura

La textura es una propiedad que representa la superficie y estructura de una imagen [4]. En líneas generales, una textura se puede definir como la repetición regular de un elemento o patrón en una superficie. Las características basadas en la textura intentan describir esos patrones. A diferencia del color, que es propiedad de un pixel individual, la textura solo puede provenir de un grupo de píxeles. Por lo general, las características basadas en texturas son potentes pero también computacionalmente costosas. Este tipo de características se pueden dividir en dos grupos. Primero tenemos los métodos basados en la *información espacial*, que extraen la información directamente de la imagen original computando estadísticas en los píxeles. Son sencillos de entender y obtienen buenos resultados en formas irregulares, aunque también son muy sensibles al ruido, la escala y las distorsiones. Por otro lado tenemos los métodos basados en *información espectral* que extraen las características de transformaciones de la imagen. Son robustos y habitualmente menos costosos.

Una de las características basadas en la información espacial de las texturas más conocida es la de Haralick, y respecto a las basadas en información espectral tenemos los filtros de Gabor, Fourier, las transformadas de Wavelet y LBP. Local Binary Pattern (LBP) codifica primitivas locales (bordes, puntos, áreas planas) en un histograma de características, y es el principal descriptor basado en texturas que se usa para la detección de personas.

### 2.2.3 Características basadas en la forma

Las características basadas en la forma codifican el tamaño y aspecto geométrico de los objetos. Las técnicas usadas para la extracción de estas propiedades se pueden dividir en dos: las que únicamente hacen uso la información localizada en el borde o frontera del objeto y las que hacen uso de la información localizada en la totalidad del área comprendida por el objeto. Las primeras tienden a ser más sensibles al ruido dado que omiten gran parte de la información del objeto al solo utilizar los bordes.

Los métodos basados en el contorno habitualmente utilizan detectores de bordes para extraer características de bajo nivel. La mayoría de estos detectores de bordes utilizan

la computación del gradiente, que además de ser la base para computar características de más alto nivel, también puede ser una por sí mismo. Algunas de las características más comunes basadas en el gradiente son: **Edge Orientation Histograms (EOH)** y **Histogram of Oriented Gradients (HOG)**. EOH define la relación entre dos orientaciones específicas en una región de la imagen, mientras que HOG consiste en contar las ocurrencias de un grupo de orientaciones en un bloque de la imagen. EOH y HOG son invariantes a los cambios de iluminación global, y HOG también lo es a las transformaciones geométricas (exceptuando la rotación). Una variación del HOG es **PHOG (Pyramid Histogram of Oriented Gradients)**, cuyo objetivo es tener en cuenta las propiedades espaciales de las formas locales; y **DOT (Domination Orientation Templates)**, que en lugar de computar los histogramas completos solo tiene en cuenta las orientaciones predominantes. **Scale Invariant Feature Transform (SIFT)** es también un descriptor basado en el gradiente, ampliamente usado para tareas de reconocimiento de objetos, por su invariabilidad a la escala y la rotación, además de su resistencia al ruido, los cambios de iluminación y las distorsiones.

#### 2.2.4 Histograma de Gradientes (HOG)

El Histograma de Gradientes Orientados (HOG) es una característica que describe la distribución de la orientación de los gradientes en cada región de la imagen [5]. Los gradientes (derivadas en x e y) de una imagen son muy útiles porque su magnitud es elevada junto a los bordes y las esquinas (regiones con cambios de intensidad importantes), y sabemos que los bordes y las esquinas contienen mucha más información de un objeto que las zonas planas. Pongamos por ejemplo que queremos detectar un botón. Un botón es circular y habitualmente tiene varios orificios para hilarlo. Puedes pasar la imagen por un detector de bordes y decidir fácilmente si es un botón mirando el resultado. La información de los bordes es útil mientras que por ejemplo el color no lo es. La idea básica es dividir la imagen en regiones más pequeñas, y para cada una de ellas, crear un histograma de gradientes con la dirección y magnitud de los mismos. Luego esos histogramas se normalizan y concatenan para obtener el descriptor final.

La implementación detallada del algoritmo es como sigue:

1. **Pre-procesamiento:** Aunque la imagen puede ser de cualquier tamaño lo habitual es utilizar imágenes con ratio 1:2, por ejemplo 64x128. En este punto también podemos realizar una normalización del color / gamma. Para ello se computa la raíz cuadrada de cada intensidad de la imagen. La normalización sólo implica un pequeño efecto en el rendimiento por lo que se puede omitir.

2. **Cálculo del gradiente:** El primer paso es calcular los gradientes horizontales ( $\partial_x(x_i, y_i)$ ) y verticales ( $\partial_y(x_i, y_i)$ ) de la imagen. Está demostrado que los mejores resultados se obtienen filtrando la imagen con los kernels  $[-1, 0, 1]$  y  $[-1, 0, 1]^T$ , sin suavizarla.



Figura 2. Imagen original, gradiente horizontal  $\partial_x(x_i, y_i)$  y gradiente vertical  $\partial_y(x_i, y_i)$ .

Con la información del gradiente horizontal y vertical, podemos calcular la orientación y magnitud para cada pixel de la imagen usando las siguientes fórmulas:

$$m(x_i, y_i) = \sqrt{\partial_x(x_i, y_i)^2 + \partial_y(x_i, y_i)^2}$$

$$\theta(x_i, y_i) = \tanh^{-1} \frac{\partial_y(x_i, y_i)}{\partial_x(x_i, y_i)}$$

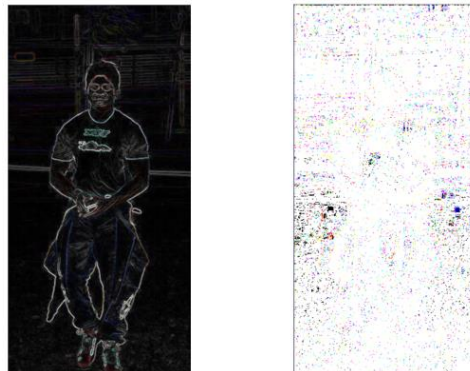


Figura 3. Magnitud ( $m$ ) y dirección ( $\theta$ ) del gradiente.

En las imágenes con color, los gradientes se calculan por separado en cada canal RGB y para cada pixel se elige el canal con mayor magnitud del gradiente.

El gradiente elimina un montón de información inútil (ej. color constante del fondo)

3. **Calculo del HOG en bloques y celdas:** La imagen se divide en regiones llamadas bloques y, al mismo tiempo, cada bloque se divide en regiones más pequeñas llamadas celdas. Se extrae un histograma por celda y se concatenan para obtener un descriptor normalizado por bloque (X-D vector). Para asegurar la consistencia en la imagen y reducir la influencia de variaciones locales se introduce una superposición entre los bloques. Es decir, el segundo bloque no empieza donde acaba el primero sino que lo hace antes, abarcando así parte de su superficie. Las celdas pueden ser rectangulares (RHOG) o circulares (CHOG). Cada histograma tiene el mismo número de *bins*, lo que determina su precisión. Los *bins* representan las orientaciones del gradiente (los ángulos) y se distribuyen de forma equitativa entre  $0^\circ$ - $180^\circ$  (gradientes sin signo) o  $0^\circ$ - $360^\circ$  (gradientes con signo). Por ejemplo, en un histograma de 9 *bins* sin signo, tendríamos en cuenta los siguientes ángulos [0, 20, 40, 60, 80, 100, 120, 140, 160]. Cada pixel de la celda contribuye al histograma añadiendo el valor de su magnitud a su correspondiente *bin*, y ese valor se llama voto. Si el ángulo de un pixel no coincide exactamente con el de uno de los *bins*, la magnitud se distribuye proporcionalmente entre los dos *bins* que lo rodean. El voto puede ser una función de la magnitud, pero se ha probado que usar directamente el propio valor produce mejores resultados.

En el siguiente ejemplo tenemos una celda de 8x8. En azul se puede apreciar que la magnitud 2 se suma al *bin* correspondiente con el ángulo de su magnitud, 80. En rojo podemos ver que la magnitud 4 se divide equitativamente entre el *bin* 0 y el 20, dado que su ángulo queda justo en medio de los dos, 10.

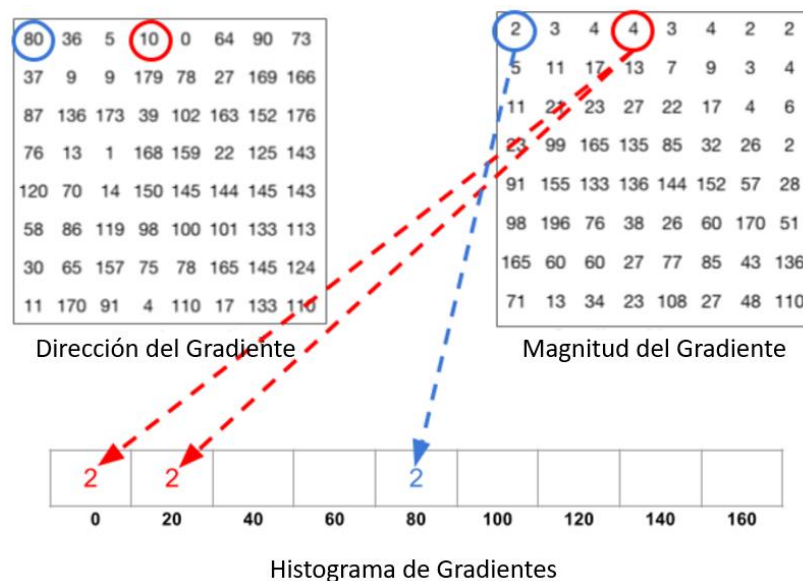


Figura 4. Ejemplo de cálculo del histograma para una celda.

La contribución de todos los píxeles de la celda 8x8 del ejemplo de arriba produciría un histograma de gradientes como este:

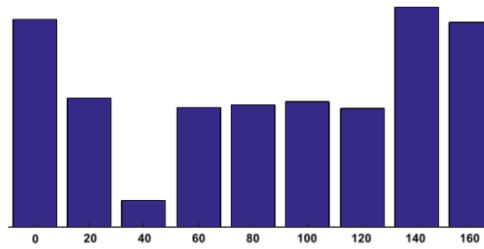


Figura 5. Histograma de gradientes correspondiente al ejemplo de la figura 4.

Esta representación de la celda no es solo mucho más compacta, también es más resistente al ruido. Como podemos apreciar en la siguiente figura, el histograma de gradientes captura perfectamente la forma de una persona.

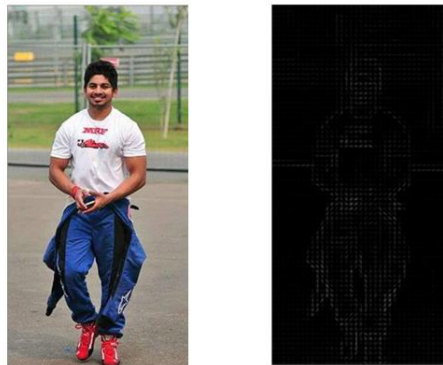


Figura 6. Imagen original y su correspondiente Histograma de Gradientes

4. **Normalización:** Es esencial para solventar problemas de iluminación. Si dividimos todos los píxeles de una imagen entre dos, la magnitud del gradiente también será dividida entre dos. Queremos que nuestro descriptor sea independiente a los cambios de iluminación. Se aplica a nivel de bloque y se pueden utilizar distintas técnicas (L1, L2...).
5. **Concatenación:** El descriptor final es un vector 1-D resultado de concatenar todos los histogramas normalizados de cada bloque de la imagen. Si por ejemplo tenemos una imagen 64x128 dividida en bloques de 16x16 (105 bloques), cada bloque tiene 4 celdas y cada celda 9 *bins*. El vector final será de dimensión  $9 \times 4 \times 105 = 3780$ .

### 2.2.5 Local Binary Pattern (LBP)

Local Binary Pattern es un descriptor basado en texturas [6]. En LBP se calcula una representación local de la textura comparando cada píxel con sus vecinos. Debido a su elevada capacidad discriminatoria, constituye una aproximación usual para la solución de multitud de problemas. Probablemente una de sus características más importantes es la robustez de su invariante ante variaciones lumínicas.

Inicialmente para el algoritmo LBP se debe trabajar únicamente con un canal de la imagen, usualmente se trabaja en escala de gris o se calcula un LBP por cada canal; se selecciona un píxel que será el eje del análisis, se determina adicionalmente un orden de comparación, el cual puede ser cualquiera que requiera el usuario, siempre y cuando, se mantenga constante en todos los análisis relacionados por esta técnica.

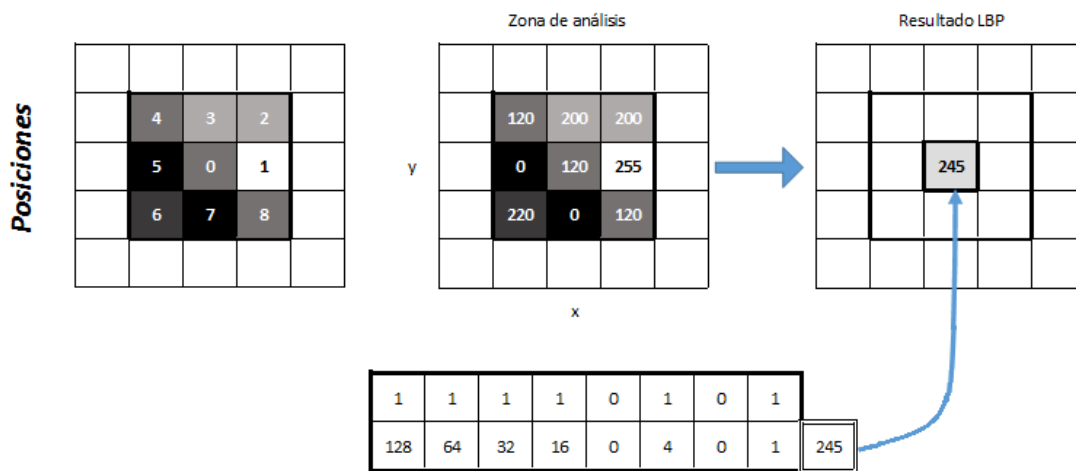


Figura 7. Cálculo de LBP.

Cuando el valor del píxel *vecino* sea igual o mayor al valor del píxel central, se asigna un valor 1 al bit de posición y en caso contrario se asigna un valor 0 hasta completar la comparación con los 8 píxeles adyacentes y formar un número binario, en el caso del ejemplo sería: 11110101. Dicho valor binario se transforma a su equivalente decimal 245 el cual corresponde al nuevo valor de intensidad de la imagen LBP.

$$f(x) = \begin{cases} 1, & \text{valor del vecino} \geq \text{valor central} \\ 0, & \text{en otro caso} \end{cases}$$

Cuando se debe realizar el análisis sobre las esquinas o los bordes de la imagen se debe definir una estrategia a usar, se puede asignar los valores ausentes como iguales a sus alrededores existentes o se puede ignorar dichas zonas y no calcular su valor LBP, depende de la aplicación necesaria pero es importante mantener constancia durante todos los LBP relacionados.

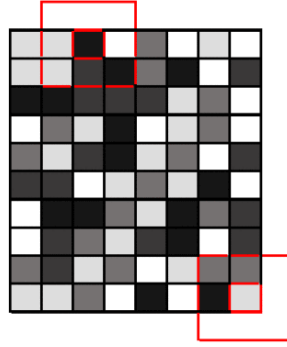


Figura 8. Tratamiento de los píxeles de los bordes y esquinas

Para mejorar los resultados se puede definir un valor umbral y asignar el estado verdadero de bit únicamente cuando la diferencia entre los niveles de gris superen o iguales los valores del umbral, tal como se define en la siguiente función.

$$f(x) = \begin{cases} 1, & \text{valor del vecino} - \text{valor central} \geq \text{umbral} \\ 0, & \text{en otro caso} \end{cases}$$

Los LBPs no se representan realmente con matrices de intensidad de color. El último paso es calcular un histograma sobre el array LBP de salida. Éste el que constituye el verdadero valor representativo, el descriptor, de la imagen seleccionada. En el caso del espacio de vecindad definido arriba (3x3), tenemos  $2^8 = 256$  posibles patrones. Esto quiere decir que nuestro array LBP tiene un valor mínimo de 0 y máximo de 255, permitiéndonos construir un histograma de 256 bins.

El principal beneficio de la vecindad 3x3 del LBP descrito es que podemos capturar detalles muy finos de la imagen. Sin embargo, esta ventaja se convierte en desventaja a la hora de capturar detalles a escalas variables, simplemente no es posible. Para contrarrestar este defecto *Ojala et al.* propuso un cambio en la implementación original de LBP. Se introducen dos parámetros nuevos.

1. El número de puntos  $p$  a considerar en una vecindad circular simétrica (dejamos de usar una vecindad rectangular).
2. El radio del círculo  $r$ , que nos permite tratar distintas escalas.

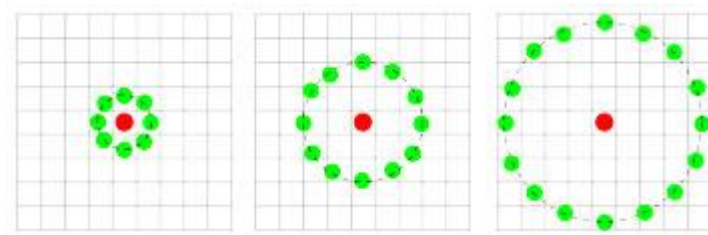


Figura 9. Tres ejemplos de vecindad con  $p$  y  $r$  variables



### 2.3 Aprendizaje máquina y clasificadores

El aprendizaje automático o de máquinas está incluido en el ámbito de las ciencias de la computación, más en concreto en el de la Inteligencia Artificial, y tiene como objetivo la creación y desarrollo de algoritmos capaces de aprender, establecer patrones sobre datos y llegar a predecirlos. De forma más concreta, se trata de crear programas capaces de aprender a resolver problemas mediante ejemplos. Es, por lo tanto, un proceso de inducción (va desde hechos, ejemplos, a afirmaciones generales) del conocimiento. En muchas ocasiones el campo de actuación del aprendizaje automático se solapa con el de la estadística, ya que las dos disciplinas se basan en el análisis de datos. Sin embargo, el aprendizaje automático se centra más en el estudio de la complejidad computacional de los problemas. Muchos problemas son de clase NP (los problemas NP son aquellos problemas para los que no se conoce un algoritmo determinista o probabilista que los resuelva en tiempo eficiente, esto es, en tiempo polinomial o menor), por lo que gran parte de la investigación realizada está enfocada al diseño de soluciones factibles a esos problemas. El aprendizaje automático puede ser visto como un intento de automatizar algunas partes del método científico mediante técnicas matemáticas.

El aprendizaje automático tiene una amplia gama de aplicaciones como pueden ser sugerencias y ayudas en motores de búsqueda, diagnósticos médicos, detección de fraude con tarjetas de crédito, detección de spam en el correo electrónico, reconocimiento de voz, traducción de idiomas mediante imágenes, detección de rostros...

No todos los sistemas de aprendizaje automático funcionan de la misma manera; en algunos se elimina toda conexión con el conocimiento que proporciona el experto acerca de los procesos de análisis de datos, mientras que otros tratan de establecer un marco de colaboración entre el experto y la máquina. De todas formas, la intuición humana no puede ser reemplazada totalmente, ya que el diseñador del sistema ha de especificar la forma de representación de los datos, los métodos de manipulación y caracterización de los mismos y por supuesto debe comprender y evaluar cómo utilizar los resultados obtenidos. Sin embargo, las computadoras son utilizadas por todo el mundo con fines tecnológicos muy útiles.

El aprendizaje automático tiene como resultado un modelo que resuelve una tarea dada. Para la producción de dicho modelo, se tienen en cuenta los atributos y características de cada ejemplo. Así, cada instancia tendrá una configuración, un dominio, esto es, un conjunto de valores que sus atributos pueden adoptar. Los valores de dicho dominio pueden ser números, valores binarios o un conjunto de etiquetas cualquiera como el de meses, estaciones o colores.

Los algoritmos de aprendizaje se clasifican en varios tipos, de acuerdo a la salida esperada del algoritmo [7].

- **Aprendizaje supervisado:**

El algoritmo genera una función que asocia las entradas del sistema con las salidas deseadas. Resumidamente utiliza conocimientos previos para obtener resultados futuros. El problema de clasificación es uno de los ejemplos más claros de aprendizaje supervisado: el sistema intenta aprender (o aproximar el comportamiento de) una función que asocia vectores de información (entradas) a una de entre varias clases (salidas) mirando ejemplos y asociaciones previas.

- **Aprendizaje no supervisado:**

Realiza el modelado únicamente a partir de entradas del sistema. No se tiene clases o salidas ni información previa, por lo que el algoritmo debe ser capaz de reconocer patrones para poder procesar correctamente datos nuevos.

- **Aprendizaje semi-supervisado:**

Toma tanto ejemplos ya clasificados como nuevos para generar una función o clasificador apropiado. Es una combinación de los dos anteriores.

- **Aprendizaje por refuerzo:**

El algoritmo aprende a actuar y modifica su comportamiento dada una observación del mundo que le rodea. Cada una de sus acciones produce un impacto en su entorno, lo que le provee un *feedback* que usa para guiar el aprendizaje. En otras palabras, el sistema aprende a base de ensayo y error.

- **Transducción:**

Es similar al aprendizaje supervisado, pero no construye de forma explícita una función. Trata de predecir nuevas salidas en base las entradas de entrenamiento, las salidas de entrenamiento, y las entradas nuevas.

- **Aprendizaje multi-tarea:**

El algoritmo aprende y realiza sus propias deducciones en base a experiencias previas.

En el presente trabajo vamos a usar principalmente algoritmos de aprendizaje supervisado.

La clasificación es el proceso en el que un elemento individual es agrupado de acuerdo a su similitud entre el propio elemento y la descripción del grupo. En otras palabras, la clasificación consiste en atribuir una categoría predefinida, etiqueta o clase a cierta instancia de un dataset separando por lo tanto los datos de entrada en categorías con características similares. Aunque los datos de entrada pueden tener muchos orígenes, en la práctica se suelen tratar como simples vectores de números.

Por lo general, la clasificación es un proceso con dos pasos:

- Fase de entrenamiento / construcción del modelo: dado un conjunto de datos de entrada (conjunto de entrenamiento) con ejemplos de las diferentes clases, el objetivo es encontrar un modelo, basado en las características extraídas de las entradas que explique el concepto de clase objetivo. El resultante modelo de clasificación, que es principalmente una función matemática asigna cada ejemplo de entrada a una de las clases predefinidas.

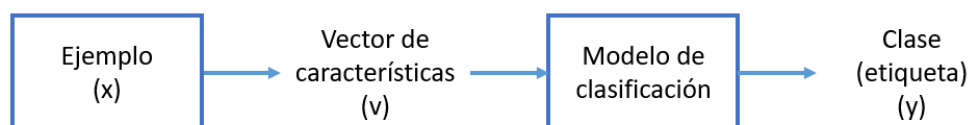


Figura 10. Tarea de asociar una instancia de entrada  $x$ , con un vector de características  $v$ , a una clase  $y$

Cada clasificador utiliza un algoritmo de aprendizaje que identifica el modelo que mejor encaja con un dataset específico.

- Fase de test / uso del modelo: El clasificador o modelo de clasificación obtenido en el paso anterior debería ser capaz de clasificar futuras, nuevas y desconocidas instancias del problema. Aun así, primero se prueba su rendimiento usando un conjunto de datos ya etiquetados, que **no** se han utilizado en el entrenamiento.

Si hablamos de clasificación en imágenes, el proceso consiste en asignar todos los píxeles o regiones de la imagen a una de las clases posibles, para de esta forma ser capaces de determinar si la imagen contiene o no un objeto particular. La detección de personas es un caso particular de clasificación binaria donde los ejemplos ( $x$ ) son imágenes y las clases pueden ser uno si la imagen contiene una persona (positivos), y cero o menos uno, si no la contiene (negativos).

A continuación se proporciona una breve descripción de algunas de las más técnicas de clasificación más representativas, y posteriormente se desarrollará y profundizará en aquellas que se vayan a utilizar en el trabajo.

- **Árbol de decisión:** Un árbol de decisión es una técnica que en función de un conjunto de atributos permite determinar a qué clase pertenece o qué valor adquiere el caso objeto de estudio. En cada nodo de decisión se especifica una prueba o test a realizar y los posibles resultados de la prueba en cuestión son los descendientes del nodo. Puede haber más de un árbol de decisión correcto para un mismo conjunto de datos dependiendo del orden en el que se van tomando los atributos. Se suele elegir cada vez el atributo que mejor clasifica, hecho que nos indica una medida denominada impureza del nodo.
- **Reglas de asociación:** Los algoritmos de reglas de asociación tratan de descubrir relaciones o patrones interesantes entre las distintas variables. Clasifican nuevos ejemplos basándose en el conocimiento aprendido. Una regla está compuesta por dos elementos; antecedente, que contiene un predicado que se evaluará como verdadero o falso para cada ejemplo, y el consecuente, que contiene una etiqueta de clase o valor numérico que adquiere el valor a predecir. A diferencia de los árboles, con las reglas no hay un orden determinado para realizar la búsqueda.
- **Máquinas de soporte vectorial:** Las Máquinas de Soporte Vectorial son una moderna y efectiva técnica de IA, aplicada fundamentalmente al procesamiento de grandes cantidades de información. Dado un conjunto de puntos, subconjunto de un conjunto mayor (espacio), en el que cada uno de ellos pertenece a una de dos posibles categorías, un algoritmo basado en SVM construye un modelo capaz de predecir si un punto nuevo (cuya categoría desconocemos) pertenece a una categoría o a la otra. La SVM busca un hiperplano que separe de forma óptima a los puntos de una clase de la de otra, que eventualmente han podido ser previamente proyectados a un espacio de dimensionalidad superior.
- **Redes neuronales artificiales:** Las redes neuronales artificiales (RNA) son un modelo de aprendizaje automático inspirado en las neuronas de los sistemas nerviosos de los animales. Se trata de un sistema de enlaces de neuronas que colaboran entre sí para producir un estímulo de salida. Las conexiones tienen pesos numéricos que se adaptan según la experiencia. De esta manera, las redes neuronales se adaptan a un impulso y son capaces de aprender. El modelo más básico del RNA se denomina perceptrón. A partir de este se pueden desarrollar sistemas más complejos denominados multicapa.

- **Algoritmo de agrupamiento:** El agrupamiento es un método de aprendizaje no supervisado y es una técnica muy popular de análisis estadístico de datos. Es un procedimiento de agrupación de una serie de vectores según criterios habitualmente de distancia; se tratará de disponer los vectores de entrada de forma que estén más cercanos aquellos que tengan características comunes.  
Ejemplos:
  - Algoritmo K-means
  - Algoritmo K-medoids
- **Redes Bayesianas:** Una red bayesiana, red de creencia o modelo acíclico dirigido es un modelo probabilístico que representa una serie de variables de azar y sus independencias condicionales a través de un grado acíclico dirigido. Una red bayesiana puede representar, por ejemplo, las relaciones probabilísticas entre enfermedades y síntomas. Dados ciertos síntomas, la red puede usarse para calcular las probabilidades de que ciertas enfermedades están presentes en un organismo. Hay algoritmos eficientes que infieren y aprenden usando este tipo de representación.
- **Regresión lineal simple y múltiple:** Es de las más utilizadas para formar relaciones entre datos ya que es rápida y eficaz. La simple es insuficiente en espacios multidimensionales donde puedan relacionarse más de dos variables. Se puede utilizar para este caso la regresión lineal múltiple.
- **Modelos estadísticos:** Es una expresión simbólica en forma de igualdad o ecuación que se emplea en todos los diseños experimentales y en la regresión para indicar los diferentes factores que modifican la variable de respuesta.

A continuación se explicará más en profundidad las técnicas y/o algoritmos utilizados en el proyecto.

### 2.3.1 Máquinas de Soporte Vectorial (SVM)

Las máquinas de soporte vectorial, introducidas por Vapnik en 1995, son una tradicional técnica de aprendizaje formal usadas para clasificación (principalmente), regresión y otras tareas, cuyo principal objetivo es optimizar un hiperplano como función de decisión que separa dos clases de datos, los ejemplos negativos y los positivos. La clasificación con menor error la consigue el hiperplano que maximiza el margen, lo que implica conseguir la mayor distancia posible entre el hiperplano y las instancias más cercanas del problema a cada uno de sus lados. Antes de maximizar el margen siempre se busca que el hiperplano separe perfectamente los ejemplos de las dos clases [8].

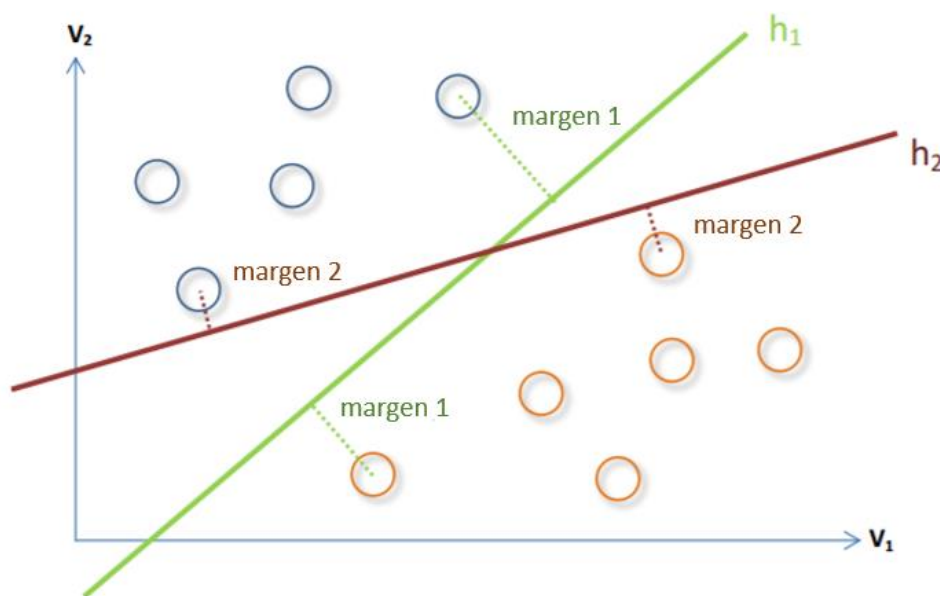


Figura 11. Representación de una SVM de dos dimensiones. Elegimos el plano  $h_1$ , que maximiza el margen.

No siempre es posible encontrar el hiperplano que separa exactamente los ejemplos de un dataset particular. El problema se puede resolver usando un margen que acepte cierto nivel de error en la clasificación.

Las SVM se usan principalmente para clasificar datos linealmente separables, pero también se pueden usar para clasificar eficientemente datos no linealmente separables aumentando el espacio dimensional del problema. La idea consiste en transformar el espacio original no linealmente separable en uno de dimensión superior, donde los datos se vuelven linealmente separables y donde es posible construir el hiperplano. La clasificación no lineal se realiza utilizando *kernels*, que son unas funciones especiales que aceptan entradas e incrementan su dimensión. Ej. Convierte problemas no separables en problemas separables. Por explicarlo de forma sencilla, realizan transformaciones complejas en los datos, y encuentra la forma de

separarlos basándose en las etiquetas y salidas que se han definido. Son especialmente útiles para la resolución de esos problemas no separables.

El buen rendimiento de las SVM se ha certificado en diversos campos como la bioinformática, el reconocimiento de imágenes, etc. Su complejidad además no se ve afectada por el número de características de los datos de entrenamiento. Por lo tanto, las SVM son adecuadas para problemas con un elevado número de características respecto al número de ejemplos.

### 2.3.2 Regresión lineal y logística

Para explicar la regresión logística, primero tenemos que explicar que es la regresión lineal. La **regresión lineal** o **ajuste lineal** es un modelo matemático usado para aproximar la relación de dependencia entre una variable  $Y$ , las variables independientes  $X_i$  y un término aleatorio  $\varepsilon$ . Este modelo puede ser representado como:

$$y_i = B_0 + B_1X_{i1} + B_2X_{i2} + \dots + B_PX_{ip} + \varepsilon_i$$

Donde:

$Y_i$ : variable dependiente.

$X_1 \dots X_p$ : Variables explicativas, independientes.

$B_1 \dots B_p$ : parámetros, miden la influencia que las variables explicativas tienen sobre la independiente. El término  $B_0$  es la intersección o término constante, las  $B_{i>0}$  son los parámetros respectivos a cada variable independiente, y  $P$  es el número de parámetros independientes a tener en cuenta en la regresión. La regresión lineal puede ser contrastada con la regresión no lineal (cuando la relación entre  $X$  e  $Y$  tiene algún grado de curvatura; algunos ejemplos son la exponencial  $Y = AX^b$  y la logarítmica  $\log(Y) = \log(A) + b \cdot \log(X)$ )

El término lineal se usa para distinguirlo del resto de técnicas de regresión, que emplean modelos basados en cualquier clase de función matemática. Los modelos lineales son una explicación simplificada de la realidad, mucho más ágiles y con un soporte teórico mucho más extenso por parte de la matemática y la estadística. La regresión lineal solo funciona por tanto correctamente cuando la relación entre variable dependiente e independientes es lineal, generan un hiperplano. Cuando solo existe una variable, la relación es una recta, si tienes más, será un plano.

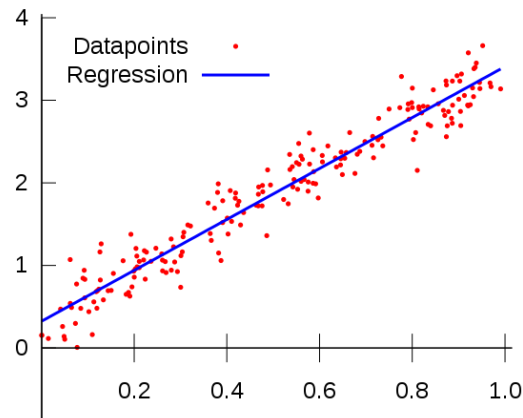


Figura 12. Ejemplo de recta de regresión sobre un conjunto de datos de una única variable

Hipótesis del modelo de regresión lineal clásico:

1. **Esperanza matemática nula:**  $E(\varepsilon_i) = 0$ . Para cada valor de X la perturbación tomará distintos valores de forma aleatoria, pero no tomará sistemáticamente valores positivos o negativos, sino que se supone tomará algunos valores mayores que 0 y otros menores que 0, de tal forma que su valor esperado sea 0.
2. **Homocedasticidad:** Varianza constante.  $Var(\varepsilon_t) = E(\varepsilon_t - E\varepsilon_t)^2 = E\varepsilon_t^2 = \sigma^2$  para todo t. Todos los términos de la perturbación tienen la misma varianza que es desconocida. La dispersión de cada  $\varepsilon_t$  en torno a su valor esperado es siempre la misma.
3. **Incorrelación o independenciam:** No existe correlación.  $Cov(\varepsilon_t, \varepsilon_s) = (\varepsilon_t - E\varepsilon_t)(\varepsilon_s - E\varepsilon_s) = E\varepsilon_t\varepsilon_s = 0$  para todo t,s con t distinto de s. Las covarianzas entre las distintas perturbaciones son nulas, lo que quiere decir que no están correlacionadas. Esto implica que el valor de la perturbación para cualquier observación muestral no viene influenciado por los valores de las perturbaciones correspondientes a otras observaciones muestrales.
4. **Regresores no estocásticos:** Regresores deterministas, que no dependen de valores aleatorios.
5. **Independencia lineal:** No existen relaciones lineales exactas entre los Regresores.
6. **T > k+1:** Suponemos que no existen errores de especificación en el modelo, ni errores de medida en las variables explicativas.
7. **Normalidad** de las perturbaciones:  $\varepsilon \sim N(0, \sigma^2)$

Existen diferentes tipos de regresión lineal que se clasifican de acuerdo a sus parámetros.

- a. **Regresión lineal simple:** cuando únicamente interviene una variable independiente en el cálculo de la variable a predecir estamos ante regresión lineal simple.



- b. Regresión lineal múltiple: La regresión lineal permite trabajar con una variable a nivel de intervalo o razón. De la misma manera, es posible analizar la relación entre dos o más variables a través de ecuaciones, lo que se denomina **regresión múltiple** o **regresión lineal múltiple**.

La **regresión logística** es un modelo clásico de regresión lineal simple o múltiple, pero donde la variable dependiente es binaria o dicotómica. Es decir, adopta sólo dos valores posibles: éxito o fracaso, positivo o negativo, objeto o no objeto. Éste tipo de regresión se utiliza para explicar y predecir una variable categórica binaria en función de varias variables independientes que a su vez pueden ser cuantitativas o cualitativas. La función de relación es una intrínsecamente no lineal. Para k variables independientes:

$$\ln\left(\frac{p}{1-p}\right) = b_0 + b_1x_1 + b_2x_2 + \dots + b_kx_k$$

### 2.3.3 K-Nearest Neighbor KNN (Vecinos más cercanos)

El KNN es sin duda uno de los algoritmos de aprendizaje formal / clasificación más sencillos. Dentro de los algoritmos de aprendizaje este entra en la clase de los supervisados. Como hemos explicado antes, la gran mayoría de métodos de clasificación empiezan induciendo el modelo clasificatorio para, posteriormente, deducir la clase de los nuevos casos. Sin embargo, en KNN ambas tareas están unidas (transducción) [9].

La idea básica sobre la que se fundamenta este paradigma es que un nuevo caso se va a clasificar en la clase más frecuente a la que pertenecen sus K vecinos más cercanos. En el caso de utilizar KNN con datos categóricos, el algoritmo devolverá la categoría a la cual debe pertenecer el caso desconocido. Si se utiliza con datos continuos, el algoritmo devolverá la media de los valores de los vecinos. El uso de KNN en clasificación se fundamenta en el uso del voto (mayoría) para decidir el valor más adecuado. Como veremos más tarde en algunas de las variaciones del algoritmo, el voto puede ser con o sin pesos. Algo a tener en cuenta en caso de tratarse de una clasificación binaria es que nos interesa un valor de la k impar para evitar empates. Veamos mejor como funciona con un ejemplo:

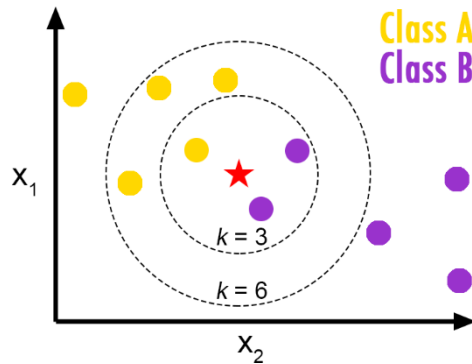


Figura 13. Ejemplo visual de KNN con dos variables y dos clases.

En la figura de arriba podemos observar que tenemos 10 ejemplos pertenecientes a dos clases distintas, 5 a la clase A en amarillo y 6 a la clase B en morado. Cada ejemplo tiene dos atributos que ayudan a clasificarlo,  $x_1$  y  $x_2$ . Como se puede observar, la distancia entre los ejemplos propios de cada clase es por lo general menor que la distancia con los ejemplos de la clase contraria, por lo que aparentan agruparse según su clase. Queremos clasificar un nuevo ejemplo, representado con la estrella roja del centro. Si elegimos  $k = 3$ , el nuevo ejemplo se clasificará con la clase B, dado que 2 de esos 3 ejemplos más cercanos pertenecen a esa clase. Sin embargo, si elegimos  $k = 6$ , el ejemplo caerá dentro de la clase A, dado que 4 de los 6 ejemplos pertenecen a esa clase. La  $k$  es por lo tanto un parámetro determinante, aunque no el único.

El algoritmo utiliza las distancias entre casos (ejemplos, vectores) para calcular los más cercanos. La elección de la métrica de la distancia también es crítica para el rendimiento del algoritmo. Entre las métricas más habituales encontramos por ejemplo la distancia Euclídea, y la de Manhattan:

$$d(p, q) = \sqrt{\sum_{i=1}^N (q_i - p_i)^2}$$

$$d(p, q) = \sum_{i=1}^N |q_i - p_i|$$

Se pueden utilizar otras métricas dependiendo del tipo de datos que se estén tratando. Para tratar distribuciones (ej. Histogramas), es habitual utilizar la distancia chi-

cuadrado. En la formula,  $n$  es el número de *bins*,  $x_i$  es un valor del primer *bin*, e  $y_i$  un valor del segundo.

$$\sum_{i=1}^n \frac{(x_i - y_i)^2}{(x_i + y_i)}$$

ó

$$\frac{1}{2} \sum_{i=1}^n \frac{(x_i - y_i)^2}{(x_i + y_i)}$$

Ahora voy a introducir muy brevemente algunas variantes sobre el algoritmo básico.

- **KNN con distancia media:** En el KNN con distancia media la idea es asignar un nuevo caso a la clase cuya distancia media sea menor.
- **KNN con distancia mínima:** En el KNN con distancia mínima se comienza seleccionando un caso por clase, normalmente el caso más cercano al baricentro de todos los elementos de dicha clase. En este paso se reduce la dimensión del fichero de casos a almacenar de  $N$  a  $m$ . A continuación se asigna el nuevo caso a la clase cuyo representante esté más cercano.
- **KNN con pesado de casos seleccionados:** La idea en el KNN con el que se efectúa un pesado de los casos seleccionados es que los  $K$  casos seleccionados no se contabilicen de igual forma, sino que se tenga en cuenta la distancia de cada caso seleccionado al nuevo caso que pretendemos seleccionar. Como ejemplo podemos convenir en pesar cada caso seleccionado de manera inversamente proporcional a la distancia del mismo al nuevo caso.
- **KNN con pesado de variables:** Otorgar la misma importancia a todas las variables, puede resultar peligroso para el paradigma KNN en el caso de que algunas de las variables sean irrelevantes para la variable clase  $C$ . Este es el motivo por el que resulta interesante el utilizar distancias entre casos que ponderen cada variable de una manera adecuada.
- **Edición de Wilson:** La idea en la edición propuesta por Wilson es el someter a prueba a cada uno de los elementos del fichero de casos inicial. Para ello, para cada caso se compara su clase verdadera con la que propone un clasificador KNN obtenido con todos los casos excepto el mismo. En el caso de que ambas clases no coincidan, el caso es eliminado. En cierto modo, el método tiene una analogía con la validación *leave-one-out*.

- **Condensación de Hart:** El condensado de Hart efectúa una selección de casos que pueden considerarse raros. Para ello, para cada caso, y siguiendo el orden en el que se encuentran almacenados los casos en el fichero, se construye un clasificador KNN con tan sólo los casos anteriores al caso en cuestión. Si el caso tiene un valor de la clase distinto al que le asignaría el clasificador K-NN, el caso es seleccionado. Si por el contrario la clase verdadera del caso coincide con la propuesta por el clasificador K-NN, el caso no se selecciona. Es claro que el método de condensación de Hart es dependiente del orden en que se encuentren almacenados los casos en el fichero.

### 2.3.4 Perceptrón Multicapa (MLP)

En primer lugar es necesario explicar qué es una red neuronal y para qué se utiliza. Las **redes de neuronas artificiales** (denominadas habitualmente como **RNA** o en inglés como **ANN**) son un paradigma de aprendizaje y procesamiento automático inspirado en la forma en que funciona el sistema nervioso de los animales. Se trata de un sistema de interconexión de neuronas que colaboran entre sí para producir un estímulo de salida. En inteligencia artificial es frecuente referirse a ellas como **redes de neuronas** o **redes neuronales**. La base de los **ANS** (Artificial Neural Systems) imita la estructura hardware del sistema nervioso, con la intención de construir sistemas de procesamiento de información paralelos, distribuidos y adaptativos, que puedan presentar un cierto comportamiento inteligente. En un sistema neuronal biológico, los elementos básicos son las neuronas, que se agrupan en conjuntos compuestos por millones de ellas organizadas en capas, constituyendo un sistema con funcionalidad propia. Un conjunto de esos subsistemas da lugar a un sistema global, el sistema nervioso. En la realización de un sistema neuronal artificial, que se organiza en capas, varias capas constituyen una red neuronal y, por último, una red neuronal junto con las interfaces de entrada y salida, más los módulos algorítmicos adicionales necesarios, conforman el sistema global de proceso.

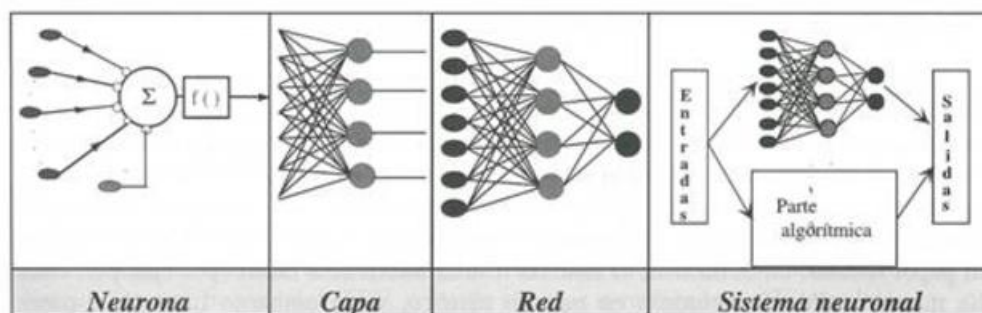


Figura 14. Estructura jerárquica de un sistema basado en ANS.

**Modelo general de neurona artificial:** Se denomina procesador elemental o neurona a un dispositivo simple de cálculo que a partir de un vector de entrada procedente del exterior o de otras neuronas, proporciona una única respuesta o salida. Los elementos que constituyen la neurona son los siguientes:

1. Conjunto de entradas  $x_j(t)$
2. Pesos sinápticos de la neurona  $i$ ,  $w_{ij}$  que representan la intensidad de interacción entre cada neurona presináptica  $j$  y la neurona postsináptica  $i$ .
3. Reglas de propagación  $\sigma(w_{ij}, x_j(t))$  que proporciona el valor del potencial postsináptico  $h_i(t) = \sigma(w_{ij}, x_j(t))$  de la neurona  $i$  en función de sus pesos y entradas.
4. Función de activación  $f_i(a_i(t-1), h_i(t))$  que indica el estado de activación actual  $a_i(t) = f_i(a_i(t-1), h_i(t))$  de la neurona  $i$ , en función de su estado actual  $a_i(t-1)$  y de su potencial postsináptico actual. Puede no existir siendo en este caso la salida de la misma función de propagación,  $a_i(t) = \sigma(w_{ij}, x_j(t))$
5. Función de salida  $F_i(a_i(t))$  que proporciona la salida actual  $y_i(t) = F_i(a_i(t))$  de la neurona  $i$  en función de su estado de activación. De este modo la activación de la neurona  $i$  puede expresarse como  $y_i(t) = F_i(f_i[a_i(t-1), \sigma(w_{ij}, x_j(t))])$

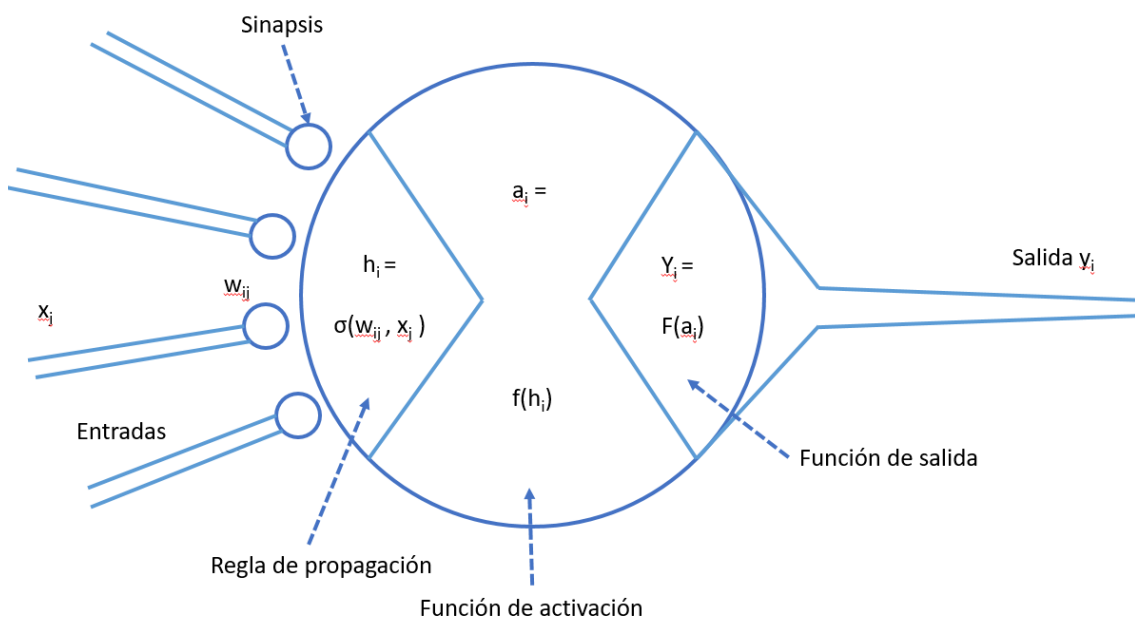


Figura 15. Estructura de una neurona artificial.

A continuación se profundiza en cada uno de los elementos que componen una neurona.

**Entradas y salidas:** Las variables de entrada y salida pueden ser binarias (digitales) o continuas (analógicas), dependiendo del modelo y aplicación. Dependiendo del tipo de salida, las neuronas suelen recibir nombres específicos. Así, las neuronas estándar cuya

salida solo puede tomar los valores 0 o 1 se suelen denominar genéricamente neuronas tipo McCulloch-Pitts, mientras que aquellas que únicamente pueden tener por salidas -1 o +1 se suelen denominar neuronas tipo Ising. Si puede adoptar diversos valores discretos en la salida (por ejemplo, -2, -1, 0, 1, 2), se dice que se trata de una neurona de tipo Potts. Por otro lado, en el caso de las analógicas puede ocurrir que una neurona de salida continua, que puede proporcionar valores cualesquiera, se limite a un intervalo definido, por ejemplo  $[0,+1]$  ó  $[-1,+1]$ .

**Regla de propagación:** La regla de propagación permite obtener, a partir de las entradas y los pesos, el valor del potencial postsináptico  $h_i$  de la neurona.

$$h_i(t) = \sigma_i(w_{ij}, x_j(t))$$

La función más habitual es de tipo lineal y se basa en la suma ponderada de las entradas con los pesos sinápticos:

$$h_i(t) = \sum w_{ij} x_j$$

El peso sináptico  $w_{ij}$  define, en este caso, la intensidad de interacción entre la neurona presináptica  $j$  y la postsináptica  $i$ . Dada una entrada positiva, si el peso es positivo tenderá a excitar a la neurona, si el peso es negativo tenderá a inhibirla.

**Función de activación o función de transferencia:** La función de activación o de transferencia proporciona el estado de activación actual  $a_i(t)$  a partir del potencial postsináptico  $h_i(t)$  y del propio estado de activación anterior  $a_i(t-1)$ .

También existen muchos modelos de ANS donde el estado actual de la neurona no depende de su estado anterior, sino únicamente del actual. La función de activación  $f()$  se suele considerar determinista, y en la mayor parte de los modelos es monótona creciente y continua. Las formas de las funciones de activación más empleadas en los ANS se muestran en la siguiente figura, donde se ha denotado con  $x$  al potencial postsináptico y con  $y$  al estado de activación. La más simple de todas es la función identidad. Otro casi también muy simple es la función escalón, empleada en el perceptrón simple.

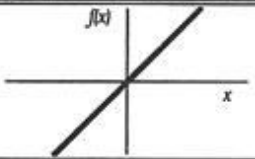
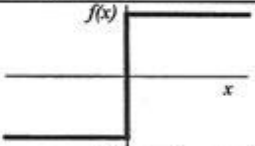
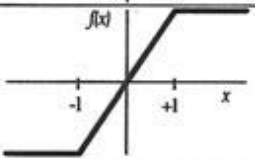
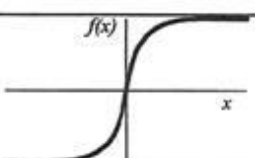
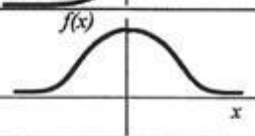
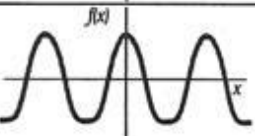
	Función	Rango	Gráfica
<b>Identidad</b>	$y = x$	$[-\infty, +\infty]$	
<b>Escalón</b>	$y = \text{sign}(x)$ $y = H(x)$	$\{-1, +1\}$ $\{0, +1\}$	
<b>Lineal a tramos</b>	$y = \begin{cases} -1, & \text{si } x < -l \\ x, & \text{si } -l \leq x \leq +l \\ +1, & \text{si } x > +l \end{cases}$	$[-1, +1]$	
<b>Sigmoidea</b>	$y = \frac{1}{1 + e^{-x}}$ $y = \text{tgh}(x)$	$[0, +1]$ $[-1, +1]$	
<b>Gaussiana</b>	$y = Ae^{-Bx^2}$	$[0, +1]$	
<b>Sinusoidal</b>	$y = A \text{sen}(\omega x + \varphi)$	$[-1, +1]$	

Figura 16. Tabla con las principales funciones de activación.

**Función de salida:** Esta función proporciona la salida global de la neurona  $y_i(t)$  en función de su estado de activación actual  $a_i(t)$ . Muy frecuentemente, la función de salida es simplemente la identidad  $F(x)=x$ , de modo que el estado de activación de la neurona se considera como la propia salida  $Y_i(t) = F_i(a_i(t)) = a_i(t)$ .

### Redes neuronales supervisadas

Uno de los modelos de redes neuronales más populares son las redes neuronales supervisadas. En este tipo de redes, con aprendizaje supervisado, se presenta a la red un conjunto de patrones, junto con la salida deseada u objetivo, e iterativamente, esta ajusta sus pesos hasta que su salida tiende a ser la deseada, utilizando para ello información detallada del error que se comete en cada paso. De este modo, la red es capaz de estimar relaciones entrada-salida sin necesidad de proponer una cierta forma funcional de partida.

Así, dentro de este grupo de redes, denominadas redes unidireccionales organizadas en capas con aprendizaje supervisado, se puede hablar del asociador lineal, perceptrón simple, adalina y perceptrón multicapa. En este último tipo habitualmente se aplica un algoritmo de aprendizaje denominado back-propagation (retropropagación) o BP.

Al añadir capas intermedias (ocultas) a un perceptrón simple, se obtiene un perceptrón multicapa o MLP (Multi-Layer Perceptron). Esta arquitectura suele entrenarse, como se ha dicho, mediante el algoritmo denominado de retropropagación de errores. Así, el conjunto arquitectura MLP + aprendizaje BP suele denominarse red de retropropagación, o simplemente BP.

Se denota  $x_i$  a las entradas de la red,  $y_i$  a las salidas de la capa oculta,  $z_k$  a las de la capa final y  $t_k$  a las salidas objetivo, además, sean  $w_{ij}$  los pesos de la capa oculta y  $\Theta_j$  sus umbrales,  $w'_{ij}$  los pesos de la capa de salida y  $\Theta'_j$  sus umbrales. La operación MPL con una capa oculta y neuronas de salida lineal se expresa matemáticamente de la siguiente manera:

$$z_k = \sum_j w'_{kj} y_j - \theta'_k = \sum_j w'_{kj} f \left( \sum_i w_{ji} x_i - \theta_j \right) - \theta'_k$$

Siendo  $f()$  de tipo sigmoide:

$$f(x) = \frac{1}{1 + e^{-x}}$$

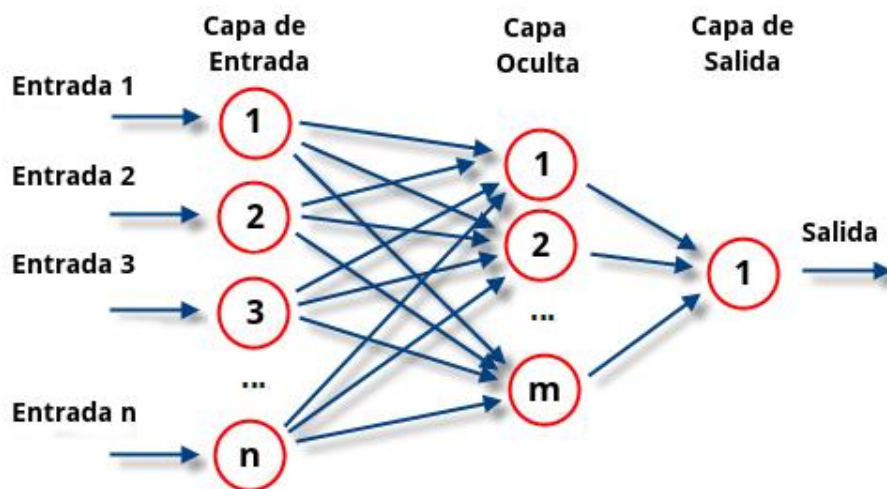


Figura 17. Estructura del perceptrón multicapa.



### 3. Implementación

En este trabajo vamos a construir un detector de personas usando varios algoritmos de clasificación (SVM, KNN, Regresión Logística, MLP), y varios tipos de características (HOG con diferentes configuraciones, y LBP). Obviamente la idea no es usar todos los clasificadores a la vez, sino utilizarlos por separado para analizar cual se comporta mejor. Parte de la implementación está basada en el trabajo de Dalal y Triggs [10].

#### 3.1 Dataset de imágenes

El dataset es uno de los elementos más importantes involucrados en el desarrollo de sistemas de aprendizaje, porque constituye la información inicial, la base sobre la que se construirá el modelo de clasificación. Debe ser lo suficientemente representativo, en términos de cantidad y calidad de los ejemplos. Si por ejemplo queremos clasificar entre perros y gatos, pero tenemos muchos menos ejemplos y de peor calidad (iluminación, ángulo...) de uno de los dos conjuntos, probablemente la eficiencia del clasificador se resentirá notablemente. El dataset también debe ser imparcial, evitando la inclusión de ejemplos peculiares que puedan alterar el resultado.

En el caso de las personas, es difícil crear un dataset general debido a la complejidad de la apariencia que pueden adquirir, afectada por las poses, iluminación, fondo, oclusión y texturas (ropa). Para desarrollar un modelo clasificatorio robusto para todas las variaciones de personas, se necesita un dataset grande y variado. Sin embargo, en ocasiones el objetivo comprende una serie de condiciones (poses predefinidas, fondos planos, sin oclusión...), y para esos casos particulares se usan datasets adaptados.

Podemos encontrar varios datasets disponibles públicamente orientados a resolver el problema de la detección de personas. Algunos de ellos solo contienen personas caminando o paradas de pie en la calle, mientras que otros incluyen un rango más amplio de posturas y perspectivas.

En este trabajo vamos a utilizar el dataset de personas **INRIA**, tanto para el entrenamiento como para el test. Es un dataset muy conocido y ampliamente utilizado para la detección de personas. En parte también elegimos este dataset para poder comparar nuestros resultados con los de otros trabajos realizados en el pasado [10]. Los ejemplos positivos tanto de entrenamiento como de test contienen imágenes de personas de pie en diferentes posiciones y sobre muy diversos fondos. Las imágenes originales son de tamaño 96x160 pero para nuestro conjunto de entrenamiento serán de 64x128, centradas en la persona y además volteadas horizontalmente para conseguir el doble de ejemplos positivos. En definitiva contamos con 2416 ejemplos de entrenamiento y 288 de test.

El conjunto de imágenes negativas de entrenamiento no contiene personas. Podemos encontrar escenas tanto interiores como exteriores. Además algunas imágenes están centradas en otros objetos como coches, bicicletas, motos, mobiliario u otros utensilios. Las imágenes originales tienen diversos tamaños, por lo general muy grandes. Nosotros utilizaremos un recorte 64x128 aleatorio de esas imágenes. El dataset se puede descargar desde <http://pascal.inrialpes.fr/data/human/>.



Figura 18. Ejemplo de imágenes positivas y negativas del dataset INRIA.

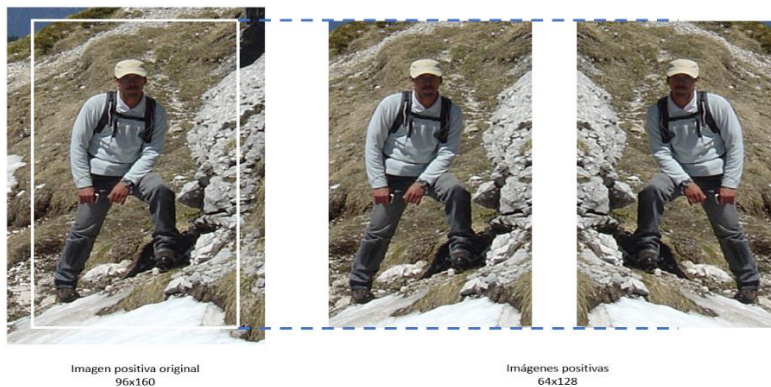


Figura 19. Representación de la generación de ejemplos positivos (Las dimensiones reales de las imágenes son aproximadas).

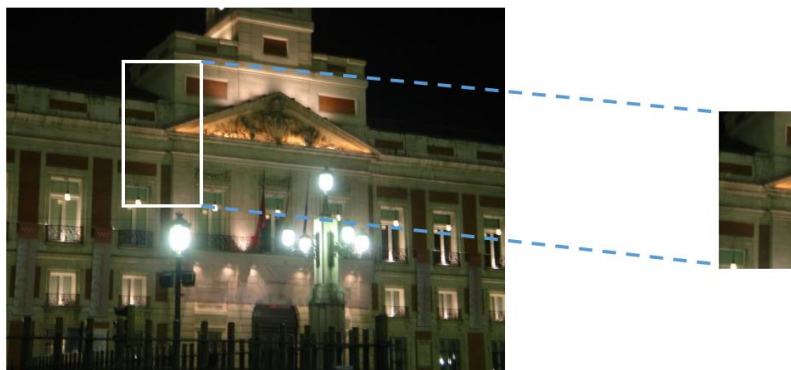


Figura 20. Representación de generación de ejemplos negativos.

### 3.2 Estructura general del algoritmo

La estructura general de un algoritmo de detección de personas se divide en dos fases. La fase de entrenamiento y la de ejecución (que se realiza por cada imagen de test). La de entrenamiento es responsable de la construcción del detector propiamente dicho, el modelo resultante de entrenar diversos clasificadores con unas características de entrada. La de ejecución consiste en evaluar nuevas imágenes con el detector construido. Debería ser capaz de detectar personas ejecutando el clasificador sobre cada región de la imagen.

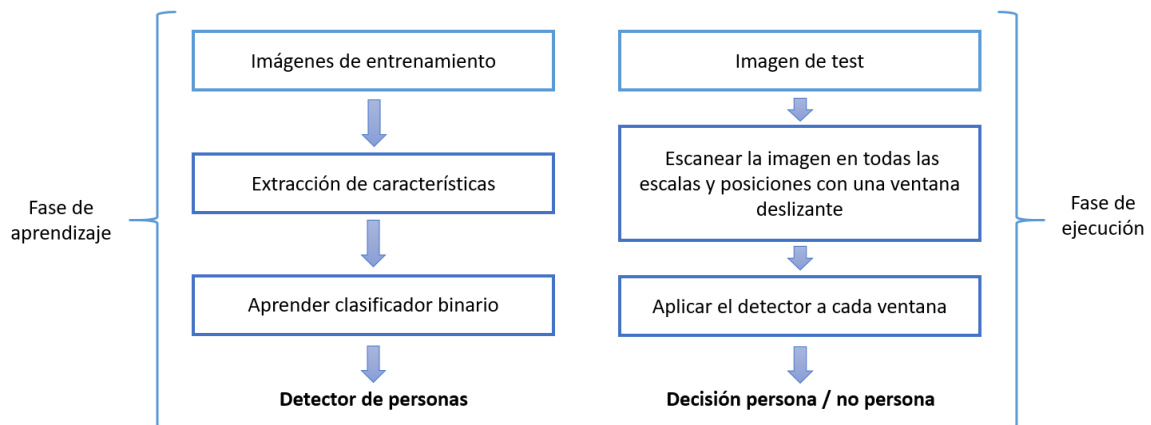


Figura 21. Estructura general del detector.

#### 3.2.1 Fase de entrenamiento

##### A. Extracción de características:

Como se ha explicado en el apartado 3.1, extraeremos las características de imágenes 64x128, todas RGB (aunque también probaremos a hacerlo en escala de grises).

Extraer las características es lo mismo que mapear la información del dataset a una dimensión fija más pequeña: los vectores de características. Se extrae un vector de cada imagen, y todos ellos forman la base de información utilizada después para entrenar el clasificador. Como características vamos a usar el Histograma de Gradientes Orientados (ver sección 2.2.4), un tipo de descriptor visual basado en el gradiente y desarrollado por Dalal y Triggs [10] para la detección de personas; y el LBP (Local Binary Patterns, ver sección 2.2.5), otro tipo de descriptor visual basado en las texturas y la estructura de la imagen desarrollado en 1994 por T. Ojala, M. Pietikäinen, y D. Harwood.

Para extraer el HOG, Dalal y Triggs, en su algoritmo, dividen cada imagen en bloques fijos de 16x16 píxeles y cada bloque contiene 4 celdas de 8x8 píxeles. Crean un histograma de gradientes con 9 bins por cada celda, y concatenando los histogramas

de las cuatro celdas, obtienen un vector de características de dimensión  $1 \times 36$  (36D) por cada bloque. Los vectores se obtienen de todos los distintos bloques posibles resultado de desplazar dicho bloque por toda la imagen (ver siguiente figura), con una superposición de 8 píxeles entre bloques adyacentes. En total, cada imagen contiene  $7 \times 15$  bloques (dado que la imagen es de  $64 \times 128$ ), lo que se traduce en un vector de características 3780D para la imagen ( $7 \times 15 \times 36$ ). El clasificador utilizará estos vectores para entrenar.

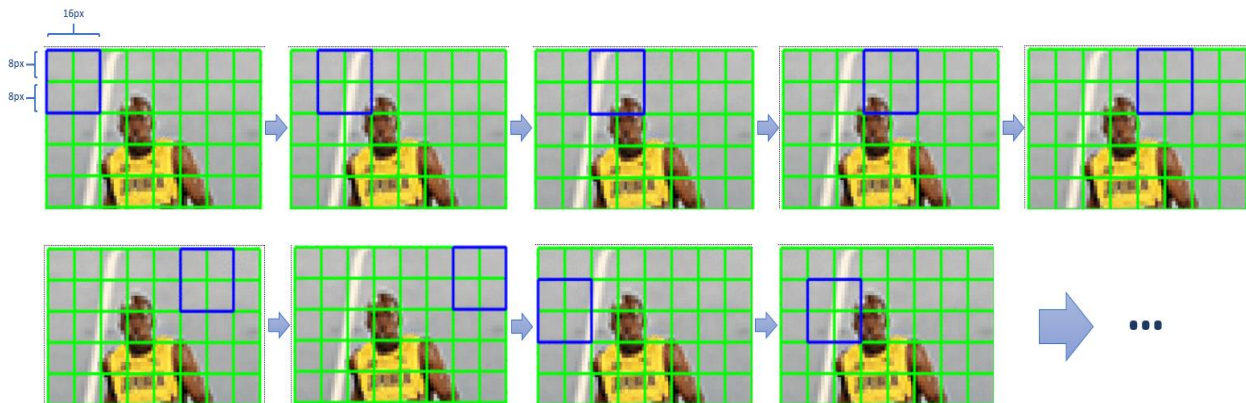


Figura 22. Movimiento del bloque a través de la ventana de detección.

Nuestra implementación está basada en la misma idea, pero además de utilizar estas medidas para el tamaño del bloque, las celdas, y los bins, también utilizaremos las siguientes para comparar resultados: Bloques de tamaño  $8 \times 8$  y  $32 \times 32$  (es decir, el doble y la mitad), manteniendo el número de celdas en cada uno (4). Al utilizar los bloques de  $8 \times 8$  obtendremos  $15 \times 31$  bloques (465), lo que se traduce en un vector 16740D ( $15 \times 31 \times 36$ ). Al utilizar los bloques de  $32 \times 32$  obtendremos  $3 \times 7$  bloques (21), lo que se traduce en un vector 756D ( $3 \times 7 \times 36$ ).

Además vamos a probar la extracción del hog normalizando y no normalizando el color (gamma) antes de computar el gradiente. Para normalizarlo aplicaremos la raíz cuadrada a cada canal de color RGB por separado.

También compararemos el rendimiento utilizando las imágenes tal y como nos vienen del dataset, a color, y transformándolas previamente a escala de grises.

Para extraer el vector de características LBP también recorreremos la imagen en “ventanas”  $16 \times 16$ , con una superposición de 8. En el caso del LBP estas ventanas serán circulares. Para LBP solo probaremos una configuración.

## B. Clasificación:

Después de extraer los vectores de características de todas las imágenes de entrenamiento, tenemos una gran lista de vectores junto con una gran lista complementaria de etiquetas que nos asocian cada vector 3780D con un valor 1 o -1 que representa persona o no persona. Estos datos son los que le pasaremos al clasificador para crear el modelo. En este proyecto, igual que con la extracción de características, vamos a probar y comparar los resultados obtenidos con más de un clasificador diferente.

El principal clasificador binario utilizado es el SVM Lineal (Máquina de soporte de vectores, ver sección 2.3.1). En los experimentos de Dalal, se ha demostrado que este clasificador es de los más precisos, fiables y escalables. Suelen ser capaces de converger de forma fiable, pueden manejar grandes datasets, y se muestran robustos a la hora de utilizar distintos vectores de características y parámetros de aprendizaje. La razón principal de usar solo la SVM Lineal, en lugar de SVM con otros Kernels, es que su tiempo de computación es notablemente inferior.

También entrenaremos varios modelos con MLP (Perceptrón multicapa, ver sección 2.3.4) con varias configuraciones de capas ocultas y neuronas; otro modelo con Regresión logística (ver sección 2.3.2), y por último varios con KNN (K Vecinos más cercanos, ver sección 2.3.3) con distintos valores para k y distintas métricas para la distancia. Todas las métricas, parámetros y opciones las veremos en la sección 3.3.2 y en los resultados.

### 3.2.2 Fase de ejecución y test

La fase de ejecución consiste en escanear concienzudamente una imagen en diferentes escalas y posiciones, mediante lo que se denomina como “pirámide de imágenes” y “ventana deslizante”.

Una “**pirámide de imágenes**” es la representación en múltiples escalas de una imagen. Utilizar una pirámide nos brinda la posibilidad de encontrar objetos de diferentes escalas en la imagen. En la parte baja de la pirámide tenemos la imagen con su tamaño original (en términos de anchura y altura). En cada capa superior, la imagen es reducida y opcionalmente suavizada (mediante un filtro gaussiano). Nosotros en este trabajo no vamos a suavizar las capas. Las capas de la pirámide se crean secuencialmente dividiendo el tamaño por un factor que podemos elegir, hasta cumplir alguna condición de parada, generalmente una restricción en el tamaño mínimo que puede tener la imagen.



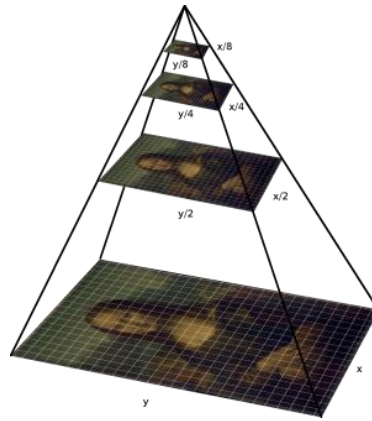


Figura 23. Ejemplo de una pirámide de imágenes. En cada capa de la pirámide la imagen se reduce y (opcionalmente) se suaviza.

La **ventana deslizante** juega un papel fundamental en la clasificación de objetos, dado que nos permite localizar exactamente “donde” está. En el contexto de la visión artificial, una ventana deslizante es una región rectangular de anchura y altura fija que se “desliza” sobre el eje horizontal y vertical de la imagen. Para cada una de estas ventanas, extraemos las características y aplicamos un clasificador para determinar si hay persona o no. El clasificador nos devuelve un valor que indica cuanto se acerca esa ventana a una clase o a otra (aunque depende del clasificador concreto). Si ese valor supera cierto umbral predefinido, podemos considerar que ahí hay una persona. La siguiente posición de la ventana se calcula sumándole un valor fijo de píxeles a la anterior en el eje correspondiente (ej. 8px), por lo que se superponen.

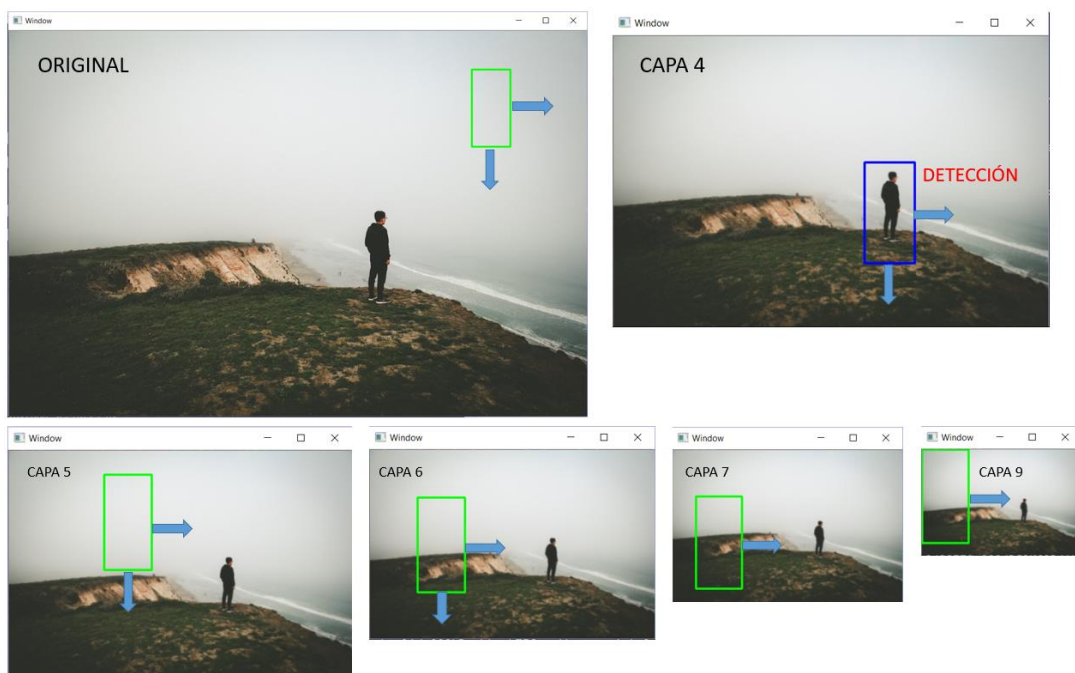
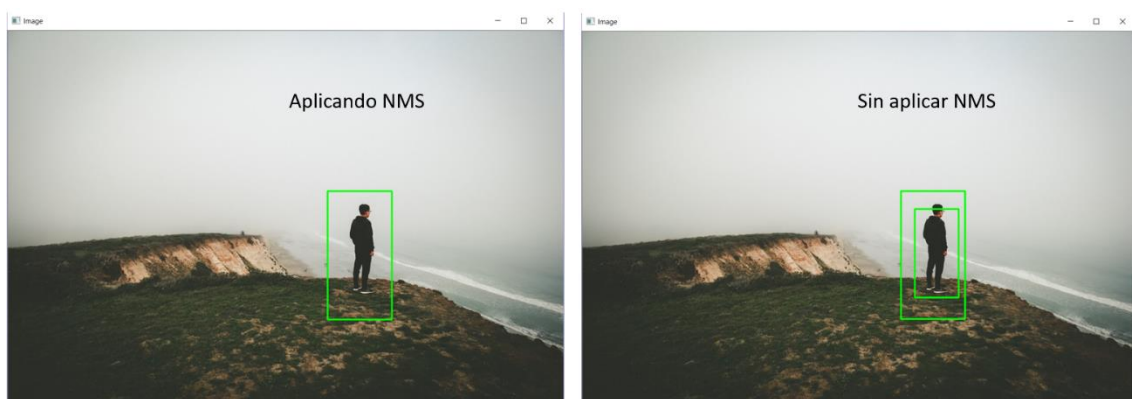


Figura 24. Ejemplo de detección mediante pirámide y ventana deslizante. Se han omitido algunas capas. Este ejemplo produce una detección en la capa 4 y otra en la 2. Ambas sobre la persona de la imagen.

Sin embargo, aunque hayamos recorrido la imagen entera en todas sus escalas y hayamos decidido en cada región, no hemos acabado. Como podemos ver en las siguientes figuras, en algunas imágenes podemos encontrarnos con dos o más detecciones sobre la misma zona, provocando un solapamiento visual de los rectángulos que indican la presencia de la persona. Para estos casos tenemos dos opciones, podemos detectar si una de las “cajas” se contiene completamente en las otras (para eliminarla), o podemos usar Non-Maxima-Suppression (NMS) para suprimir los rectángulos que se solapan con cierto margen. La implementación utilizada para NMS es la de Malisiewicz, que es sobre 100 veces más rápida que la de Felzenszwalb, su predecesora.



*Figura 25. Resultado de aplicar o no NMS a las detecciones positivas al final del proceso. A la derecha vemos que en dos escalas diferentes se ha detectado a la misma persona, lo que produce dos positivos cuando solo debería ser uno.*

Por último, el test del detector consiste en utilizar en la fase de ejecución todas y cada una de las imágenes de test, que recordemos, no hemos utilizado previamente para entrenar. Con los resultados obtenidos podemos realizar un análisis estadístico, y comparar clasificadores en base a su precisión y eficiencia. Los resultados y análisis los veremos en detalle en la sección 4.

### 3.3 Desarrollo general del algoritmo

#### 3.3.1 Entorno de desarrollo

El algoritmo se ha implementado desde cero en **Python 3.6** usando un ordenador con Windows 10. Python es un lenguaje de programación multiparadigma, ya que soporta orientación a objetos, programación imperativa y, en menor medida, programación funcional. Es un lenguaje interpretado, usa tipado dinámico y es multiplataforma. Su filosofía hace hincapié en una sintaxis que favorezca un código legible. Python destaca entre los lenguajes de programación interpretados por su amplia y activa comunidad dedicada a la computación científica. El uso de Python para tareas de computación científica se ha incrementado notablemente desde principio de siglo tanto en aplicaciones industriales como en investigación académica. Al realizar análisis, exploración, y visualización de datos es inevitable comparar Python con otros lenguajes de código abierto o comerciales ampliamente conocidos como R, MATLAB, SAS, Stata y otros. En los últimos años, el incremento en la calidad, variedad y soporte de las librerías científicas disponibles para Python, lo han convertido en un lenguaje muy a tener en cuenta para tareas de manipulación de datos. Combinando eso con su potencial en términos generales de programación, su simplicidad, y su portabilidad hace de Python una excelente elección para construir aplicaciones centradas en los datos. Librerías utilizadas en el proyecto, como OpenCV tienen el núcleo de sus cálculos programados y compilados en lenguajes más rápidos y eficientes como C++, y simplemente exponen sus funciones a Python, que se beneficia de esa velocidad.

Se ha usado GitHub, un servicio de hosting web para desarrollo de software, para guardar el código del proyecto. La última versión del código está disponible en <https://github.com/soliddanii/HumanDetector>.

Las simulaciones se han ejecutado sobre una máquina con las características de la siguiente figura.

<b>Sistema Operativo:</b>	Windows 10 Pro N Sistema de 64 bits Build 14393
<b>Hardware:</b>	Pro: Intel Core i5-3210M @ 2,5GHz RAM: 6.00 GB (5,86 Utilizables)

*Figura 26. Especificación del ordenador utilizado para ejecutar el algoritmo.*



### 3.3.2 Estructura del código

El código está organizado en distintas funciones de Python. Cada una de ellas tiene un cometido en particular, que describiremos en esta sección, junto con los parámetros, y librerías utilizadas.

#### ❖ Función **extractFeatures**:

Durante la ejecución de todo el algoritmo nos servimos de un conjunto de variables globales de configuración donde están guardados los parámetros y los directorios del dataset que vamos a utilizar. Esta función no recibe ningún parámetro de entrada, y es la primera que se ejecuta al iniciar el algoritmo. El primer cometido es crear una carpeta por cada subconjunto del dataset (ejemplos positivos y negativos). En estas dos carpetas guardaremos los vectores de características de cada subconjunto respectivamente. El segundo paso es llamar a la función *extractAndStoreFeatures()*, primero con los ejemplos positivos y después con los negativos.

#### ❖ Función **extractAndStoreFeatures**:

Una de las peculiaridades del algoritmo que hemos implementado es que no necesariamente tiene que ejecutar siempre todos los pasos. Para entrenar el clasificador bastará con extraer los vectores de características del dataset de entrenamiento una sola vez (con cada conjunto de parámetros elegido obviamente). Esta función recibe como parámetros el directorio del dataset que se va a utilizar, y el directorio donde se van a guardar los vectores extraídos.

Por cada imagen .png que encontremos en el directorio:

1. Comprobamos que no hemos calculado y guardado ya su vector de características. Si se da el caso simplemente saltamos el ejemplo.
2. Cargamos la imagen mediante la función *imread* de OpenCV. Esta función nos permite decidir si queremos cargar la imagen con sus tres componentes de color o solo en escala de grises.
3. Extraemos las características.
  - a. Si lo que nos interesa es sacar el hog, llamaremos a la función *extractHOGfeatures()* con la imagen como parámetro.
  - b. Si por el contrario queremos extraer el LBP, llamaremos a la función *extractLBPfeatures()* también con la imagen como parámetro.
4. Guardamos el vector de características que nos ha devuelto la función llamada en el paso anterior. Para guardarlo utilizamos la función *dump* de la librería *joblib*, dado que se ha demostrado que es más rápida y eficiente que *pickle* a la hora de guardar estructuras de *numpy*.

#### ❖ Función **extractHOGfeatures**:

Esta función crea un descriptor hog de OpenCV con los parámetros correspondientes. Después llama a la función `compute` del descriptor y devuelve el vector calculado. A parte de los parámetros que ya hemos comentado en la sección 3, también tenemos que usar otros que no variamos nunca, como la forma en la que tratar el ángulo del gradiente (sin signo), o el tipo de normalización de bloques utilizada (L2-Hys).

En algún momento del desarrollo del trabajo también probamos a calcular el hog con la función correspondiente de la librería SKimage, pero descubrimos que era considerablemente más lenta haciendo los cálculos que OpenCV, además de solo permitir como entrada imágenes en escala de grises.

#### ❖ Función **extractLBPfeatures**:

Esta función lo primero que hace es convertir la imagen de entrada de color a escala de grises, dado que LBP se utiliza sobre imágenes de una sola dimensión. Después crea un descriptor LBP con la ayuda de la función `local_binary_pattern()` de `skimage`. El descriptor toma como parámetros, la imagen, el método LBP, el radio y la cantidad de puntos. Después se divide la imagen en cada ventana LBP posible y se calculan sus histogramas para juntos formar el vector de características LBP.

#### ❖ Función **trainModel**:

En esta función se entrenan los modelos o clasificadores de acuerdo a los parámetros globales. Lo primero que hace es comprobar que el archivo del modelo no existe ya, si se da el caso finaliza directamente (Se puede saltar la comprobación y forzar a reentrenar pasándole el parámetro `-t` al programa). Después, lee cuantos ejemplos tenemos (archivos `.feat` que contienen las características de un ejemplo), y de qué tamaño son los vectores de características para inicializar un array “X” de numpy donde cargarlos. Por cada archivo de características, se carga y se guarda en su correspondiente fila de la matriz. Luego para las etiquetas, tan solo tenemos que crear un vector “y” de longitud la cantidad total de vectores, con tantos 1 como ejemplos positivos, y tantos -1 como ejemplos negativos.

El siguiente paso consiste en entrenar el clasificador propiamente dicho y la función utilizada depende del elegido, aunque todas provienen de la librería `sklearn`.

Para definir un clasificador SVM lineal se usa la función `LinearSVC()` con el parámetro variable `C`, que hace referencia a la penalización del error en el algoritmo.

Para definir un clasificador MLP se usa la función `MLPClassifier()` con los parámetros variables `activation` (que hace referencia a la función de activación de la capa oculta), y `solver` (que hace referencia al método de optimización de pesos). Según la documentación de `sklearn`, el resolvidor `lbfgs` puede converger mejor y más rápido

con datasets pequeños, y el resolvidor *adam* funciona mejor con datasets grandes. Respecto a las capas y neuronas hemos decidido optar por una capa de entrada con muchas neuronas (480), dado que tenemos muchas características, por dos capas ocultas con 220 y 50 neuronas respectivamente, y por una capa de salida con 2 neuronas.

Para definir un clasificador de regresión logística se usa la función *LogisticRegression()*. Para este modelo hemos decidido no variar los parámetros.

Para definir un clasificador KNN se usa la función *KNeighborsClassifier()* con los parámetros variables *k* (número de vecinos) y *metric* (métrica para calcular la distancia entre vecinos, en nuestro caso Euclídea o de Manhattan).

Por último, se llama a la función *fit(X, y)* sobre el modelo definido para entrenarlo, se obtiene su precisión sobre los propios ejemplos de entrenamiento con la función *score(X, y)*, y se guarda en un fichero de nuevo con *dump()* de joblib.

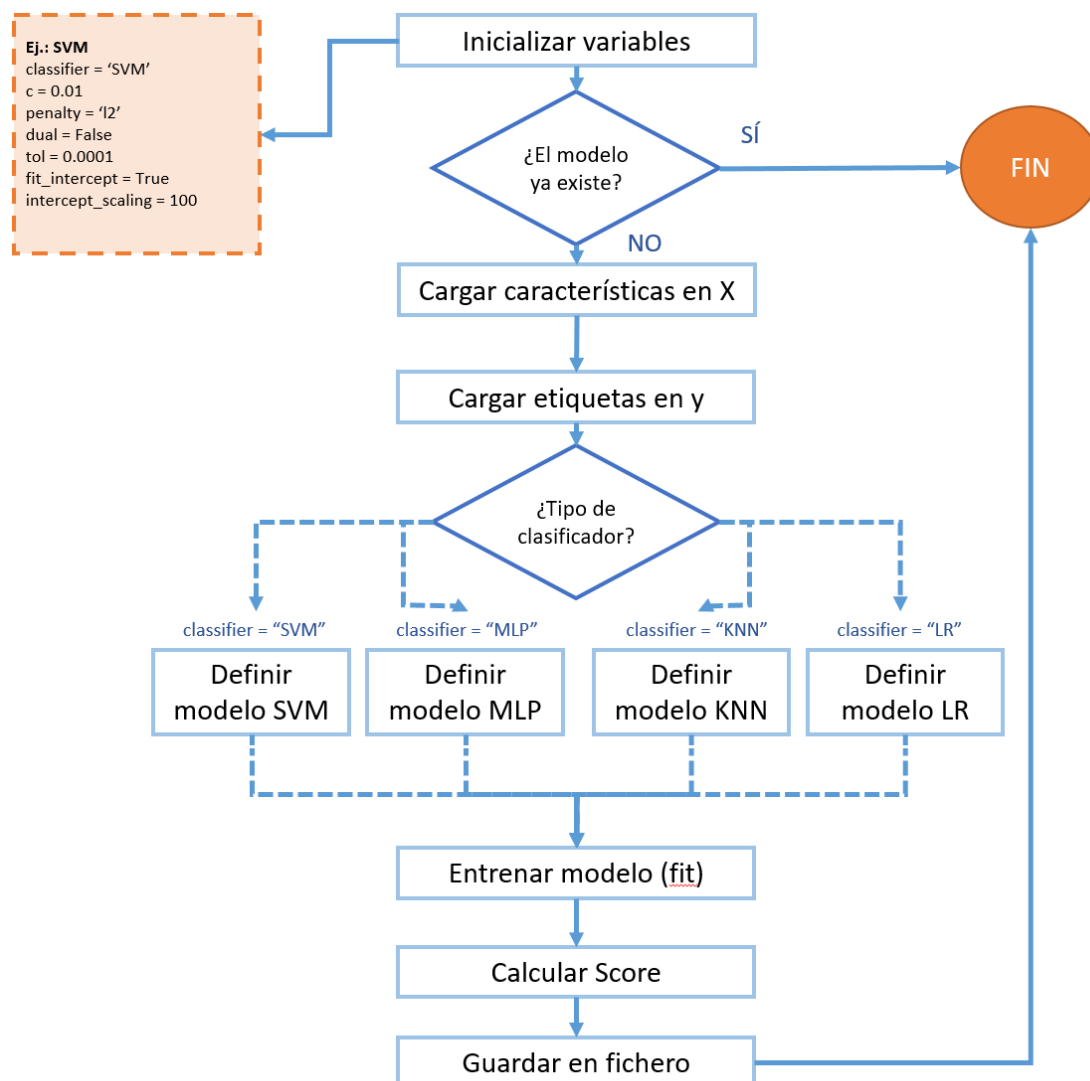


Figura 27. Esquema de ejecución de la función *trainModel()*

#### ❖ Función **testFolder**:

Esta función se utiliza para testear todas las imágenes de un directorio con un determinado clasificador. Por cada imagen analizada, se guarda un fichero con la posición de las personas que se han detectado.

Por cada imagen .png que encontremos en el directorio de test:

1. Se llama a la función *testImage()* que analiza la imagen y devuelve las tuplas que representan la posición, tamaño y score de una persona (grado de seguridad con el que se considera que ahí hay una persona).
2. Se captura también el tiempo que se ha tardado en analizar la imagen.
3. Se crea una estructura de diccionario donde se cargan los datos que tenemos hasta ahora, y finalmente se guarda en un fichero.

#### ❖ Función **testImage**:

Primero se carga el modelo que se va a utilizar para predecir la imagen. Después se carga la propia imagen y se inicializan las variables donde se van a guardar las personas detectadas. El resto consiste en recorrer un bucle doble. El exterior recorre la pirámide de imágenes con ayuda de la función *pyramid()*, el interior recorre cada capa de la pirámide con una ventana deslizante con ayuda de la función *sliding\_window()*. Dentro del bucle se dispone de una variable llamada *window* que hace referencia a la ventana deslizante concreta sobre la que se trabaja.

Por cada ventana deslizante *window*:

1. Extraer las características de la ventana.
2. Predecir la clase con las características extraídas y la función de predicción correspondiente al clasificador que estemos utilizando (*predict\_proba()* para SVM o *decision\_function()* para MLP, por ejemplo).
  - a. Si se detecta persona, se guarda la posición de la ventana y el score de predicción en las variables que se han inicializado al principio.
  - b. Si no se detecta persona, pasamos a la siguiente ventana.

#### ❖ Función **nonMaxSuppression** y **readInriaAnnotations**:

La primera función sirve para aplicar el algoritmo Non Max Suppression al conjunto de rectángulos resultado de las detecciones realizadas por *testImage()*. Los rectángulos que se solapan con cierto margen se reducen a uno solo. La segunda sirve para leer las anotaciones del dataset de test. Una anotación es un fichero donde en cada línea aparece una tupla de cuatro números que forman las coordenadas de una ventana donde aparece una persona en la imagen. Por cada imagen de test positiva hay un fichero de anotación.

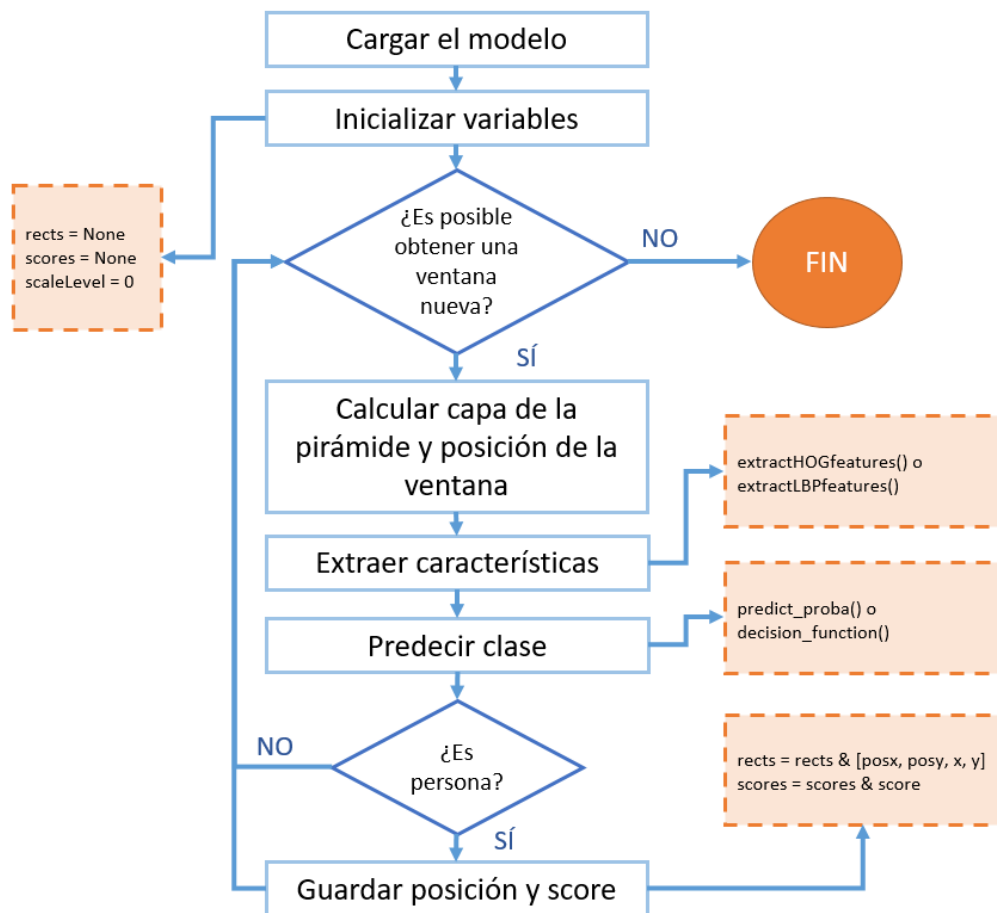


Figura 28. Esquema de ejecución de la función `testImage()`

#### ❖ Función **evaluate**:

Esta es la función que se ejecuta en último lugar, y que utiliza los resultados producidos por `testFolder()` para realizar estadísticas. Primero inicializamos los vectores donde cargar los falsos positivos (ej. `FP[0]` serían los falsos positivos de la primera imagen), las personas detectadas, las personas no detectadas, la cantidad de ventanas utilizadas y el tiempo tardado en analizar la imagen.

Después, por cada archivo en el directorio de resultados:

1. Cargamos los datos mencionados arriba.
2. Cargamos el archivo de anotaciones si existe.
3. Calculamos las estadísticas. Por ejemplo, si es una imagen negativa porque no existe su correspondiente archivo de anotaciones, y tiene 3 detecciones, asignamos tres a los falsos positivos producidos por esta imagen. Consideramos que una imagen se ha analizado con éxito si no tiene falsos positivos, y tiene tantos verdaderos positivos como personas. Con los falsos y los verdaderos positivos y negativos podemos computar estadísticas como FPPW (False Positives per Window) y FPPI (False Positives per Image).

## 4. Resultados

### 4.1 Métricas y parámetros

El objetivo de esta sección es aclarar algunas de las métricas y parámetros que vamos a utilizar, además de definir una configuración de clasificación base con la que entrenar y probar el detector. El rendimiento obtenido de este clasificador base se va a utilizar como referencia para comparaciones posteriores.

- **Factor de escala:** factor por el que se divide el tamaño de la imagen para obtener la siguiente capa de la pirámide.
- **Umbral de decisión:** Valor que debe superar la puntuación de una clasificación para que la ventana clasificada se considere como persona.
- **Número de ventanas –  $W$ :** Cantidad de ventanas deslizantes utilizadas para analizar una imagen. Como ya hemos visto, depende del tamaño original de la imagen, del parámetro factor de escala, y del tamaño mínimo de la imagen.
- **Número de imágenes –  $N$ :** Cantidad de imágenes escaneadas.
- **Verdaderos positivos –  $TP$ :** Cantidad de ventanas detectadas correctamente como persona.
- **Verdaderos negativos –  $TN$ :** Cantidad de ventanas detectadas correctamente como “fondo”.
- **Falsos positivos –  $FP$ :** Cantidad de ventanas detectadas como persona cuando no lo eran.
- **Falsos negativos –  $FN$ :** Cantidad de ventanas detectadas como “fondo” cuando en realidad eran personas.
- **True Positive Rate –  $TPR$ :**  $TP/(TP+FN)$ . Proporción de verdaderos positivos que se predicen como positivos. Es decir, probabilidad de detección. También llamado Sensitivity.
- **False Positive Rate –  $FPR$ :**  $FP/(FP+TN)$ . Proporción de verdaderos negativos que se predicen como positivos. Es decir, probabilidad de falsa alarma. También llamado fall-out.
- **Falsos positivos por ventana –  $FPPW$ :** Número total de falsos positivos del test entre número total de ventanas del test.
- **Falsos positivos por imagen –  $FPPI$ :** Número total de falsos positivos del test entre número total imágenes del test.
- **Accuracy:**  $TP+TN / \text{Total}$ .

- **Tiempo medio de detección:** Tiempo medio que tarda el detector en escanear una imagen.
- **Imagen escaneada satisfactoriamente:** Imagen con 0 FP y 0FN.
- **Porcentaje de éxito:** Cantidad de imágenes escaneadas satisfactoriamente entre cantidad de imágenes total.

#### CLASIFICADOR BASE

Dataset	2990 Negativos 2416 Positivos
Vector de características	Tipo: HOG Tamaño del bloque: 16x16 Dimensión de la imagen: RGB Corrección de color: NO
Clasificador	Tipo: SVM C = 0.01
Detector	Umbral de decisión: 0.3 Factor de escala: 1.2

Figura 29. Parámetros del clasificador de referencia o base.

## 4.2 Resultados del clasificador base

### Resultados

Cantidad de Imágenes y Ventanas	
N	588
W	309850
Rendimiento - Aciertos y Fallos	
TP	448
TN	308940
FP	321
FN	141
Rendimiento en Tiempo (ms)	
Tiempo Medio	274.71
Tiempo Medio Positivo	426.05
Tiempo Medio Negativo	129.43
Éxito General	
Éxito	342
% Éxito	58.16%

Tabla 1. Resultados del clasificador base.

En la tabla de arriba podemos ver los resultados obtenidos por el clasificador base sobre las 588 imágenes de test. En líneas generales, se puede decir que se han clasificado correctamente el 58.16% de los ejemplos. Con los datos de aciertos y fallos, podemos computar las estadísticas mencionadas en la sección anterior.

$$\text{True Positive Rate} = \text{TPR} = \frac{TP}{TP + FN} = \frac{448}{448 + 141} = 76.06\%$$

$$\text{False Positive Rate} = \text{FPR} = \frac{FP}{FP + TN} = \frac{321}{321 + 308940} = 0.1038\%$$

$$\text{FPPW} = \frac{FP}{W} = \frac{321}{309850} = 1.03599 \times 10^{-3}$$

$$\text{FPPI} = \frac{FP}{N} = \frac{321}{588} = 0.5459$$

$$\text{ACCURACY} = \frac{TP + TN}{TP + TN + FP + FN} = \frac{448 + 308940}{448 + 308940 + 321 + 141} = 0.9985$$

Como podemos apreciar, la probabilidad de detectar una persona cada ventana es del **76.06%**, y la de tener una falsa alarma es del **0.1038%**. Sin embargo, si hablamos en términos de imágenes en lugar de ventanas, tenemos más o menos un falso positivo por cada dos imágenes escaneadas. La cantidad de los falsos positivos es relativamente elevado y una de las causas de que el éxito en la detección no llegue al 60% a pesar de tener un TPR del 76%. En las comparaciones que vamos a realizar a continuación incluiremos éstos datos directamente en la tabla.

El tiempo medio de detección por imagen es más o menos 250ms, aunque si lo desglosamos en imágenes positivas y negativas, las positivas consumen bastante más tiempo superando los 425ms. El tamaño de las imágenes de test es variable desde 1200x900 hasta 400x320 (aproximadamente), y podemos encontrar más imágenes grandes en el conjunto de ejemplos positivos, lo que explicaría en cierto grado esa variación en el tiempo de ejecución.



### 4.3 Umbral de detección

Lo primero que vamos a comprobar es como varía el número de TP y FP cuando se sube o se baja el valor del umbral de detección. La ventana tiene que superarlo para ser considerada una persona, por lo que cuanto más alto sea el umbral menos falsos positivos encontraremos, pero también detectaremos menos personas. El objetivo es encontrar un punto de equilibrio. Para analizar este parámetro vamos a dibujar la curva ROC (Característica Operativa del Receptor).

La curva ROC es una representación gráfica de la sensibilidad frente a la especificidad para un sistema clasificador binario según se varía el umbral de discriminación. O en otras palabras, es la representación de la razón o ratio de verdaderos positivos (TPR = Razón de Verdaderos Positivos) frente a la razón o ratio de falsos positivos (FPR = Razón de Falsos Positivos) también según se varía el umbral de discriminación.



Figura 30. Curva ROC. En rojo aparece el punto correspondiente al umbral base 0.3.

Esta curva ilustra el rendimiento general de nuestro clasificador binario. Como podemos ver, aún podríamos aumentar ligeramente el umbral hasta 0.4 o 0.5 para ganar TP sin introducir una gran cantidad de FP. En las siguientes gráficas podemos observar como varían los valores de la tasa general de éxito, las detecciones y los falsos positivos según el umbral. Para valores de aproximadamente 0.2 hacia abajo la tasa de falsos positivos se dispara exponencialmente mientras que la de detecciones se mantiene bastante estable. A partir de aproximadamente 0.3-0.4 en adelante ambas cantidades decrecen hasta estabilizarse.

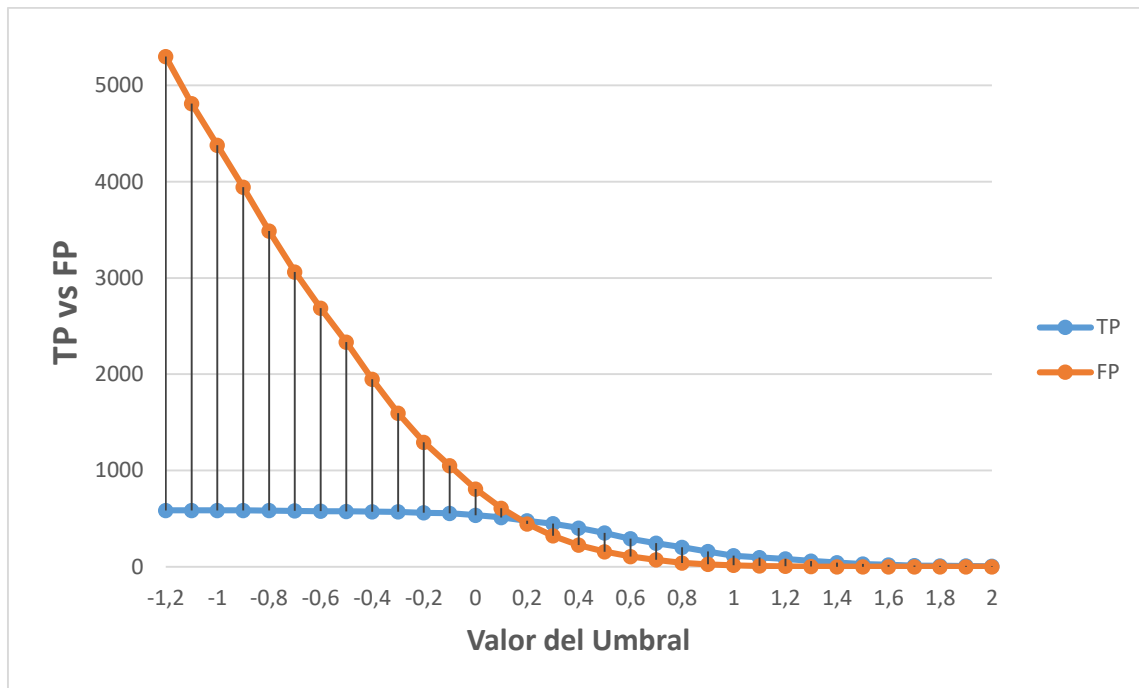


Figura 31. Valor de TP y FP según el umbral escogido.

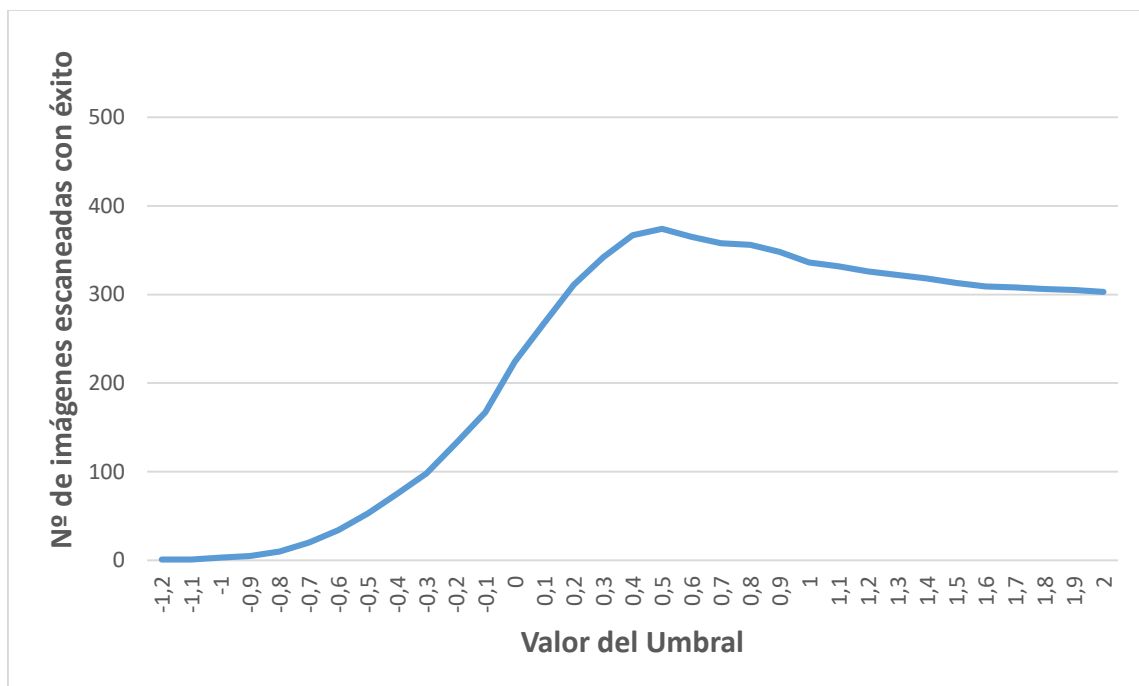


Figura 32. Cantidad de imágenes escaneadas con éxito según el Umbral. El punto más alto se corresponde con un umbral de 0.5, 374 imágenes escaneadas exitosamente. Un 63.61%.

Como veremos más adelante en la comparación de resultados, el valor del umbral para conseguir un ratio de detección aceptable sin disparar el de falsos positivos varía con cada clasificador.

#### 4.4 Factor de escala

Otro parámetro del detector que podemos analizar es el factor de escala. Recordemos que este factor dicta cuanto debe reducirse el tamaño de una imagen entre cada capa de la pirámide de imágenes. Un factor de escala 1.0 supondría una reducción nula y por lo tanto la imagen nunca acabaría de analizarse. A partir de 1.0, cuanto mayor sea el factor más pequeña será la siguiente capa, menos capas habrá, porque el tamaño de la imagen alcanzará antes el mínimo, y por lo tanto, menos tiempo de computación consumirá el detector. Una menor cantidad de capas en la pirámide también tiene otra consecuencia. La probabilidad de que el tamaño de la ventana de detección coincida con el tamaño de la persona que se haya en la imagen también disminuye.

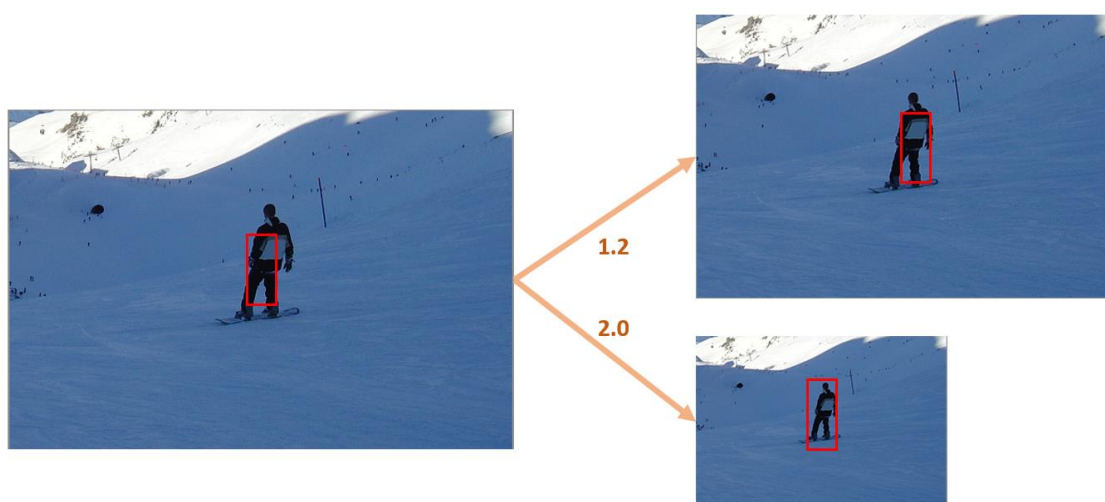


Figura 33. Escalado de la imagen. En este ejemplo un factor de escala 2 aumentaría las posibilidades de hacer coincidir la ventana deslizante con la persona.

Factor de Escala	1.1	BASE	1.5	2.0	2.5	3.0	4.0
<b>Resultados</b>							
<b>TP</b>	524	448	299	195	184	156	141
<b>FP</b>	469	321	197	164	155	151	150
<b>Tiempo medio de Test (ms)</b>							
<b>Imagen</b>	480.10	274.71	176.90	128.45	120.19	114.21	107.62
<b>I. Positiva</b>	754.01	426.05	275.77	199.44	184.24	174.32	163.90
<b>I. Negativa</b>	217.14	129.43	81.98	60.31	58.70	56.51	53.59
<b>Precisión</b>							
<b>FPPI</b>	0.7976	0.5459	0.3350	0.2789	0.2636	0.2568	0.2551
<b>FPPW</b>	0.0009	0.0010	0.0011	0.0011	0.0012	0.0012	0.0012
<b>TPR</b>	88.96%	76.06%	50.76%	33.11%	31.24%	26.49%	23.94%
<b>FPR</b>	0.09%	0.10%	0.11%	0.11%	0.12%	0.12%	0.12%

Tabla 2. Resultados de los distintos factores de escala. El factor base es 1.2.

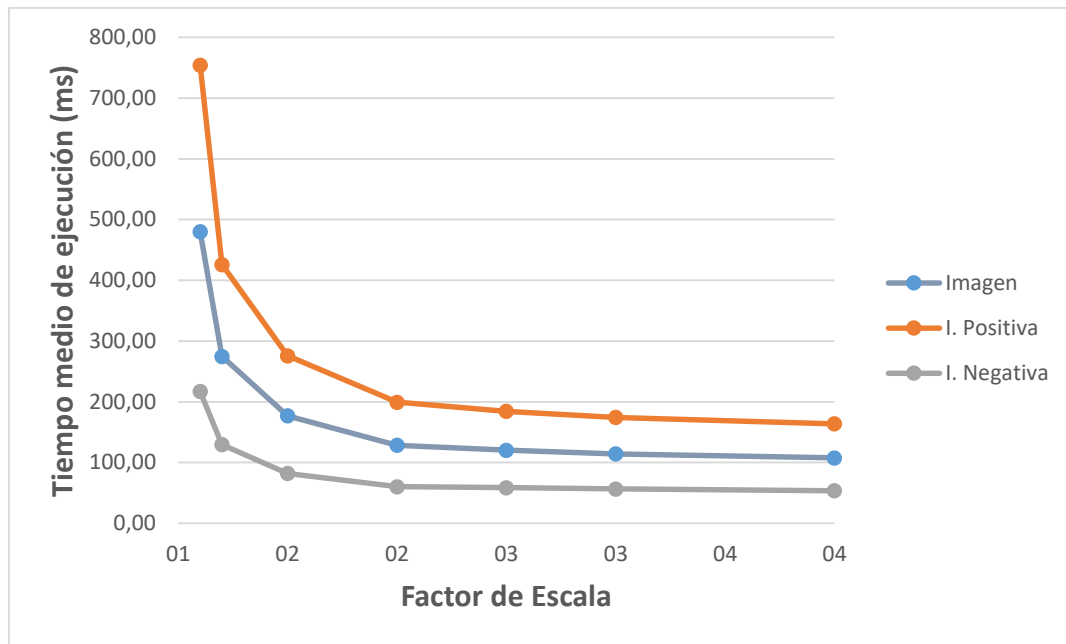


Figura 34. Tiempo de ejecución respecto al factor de escala.

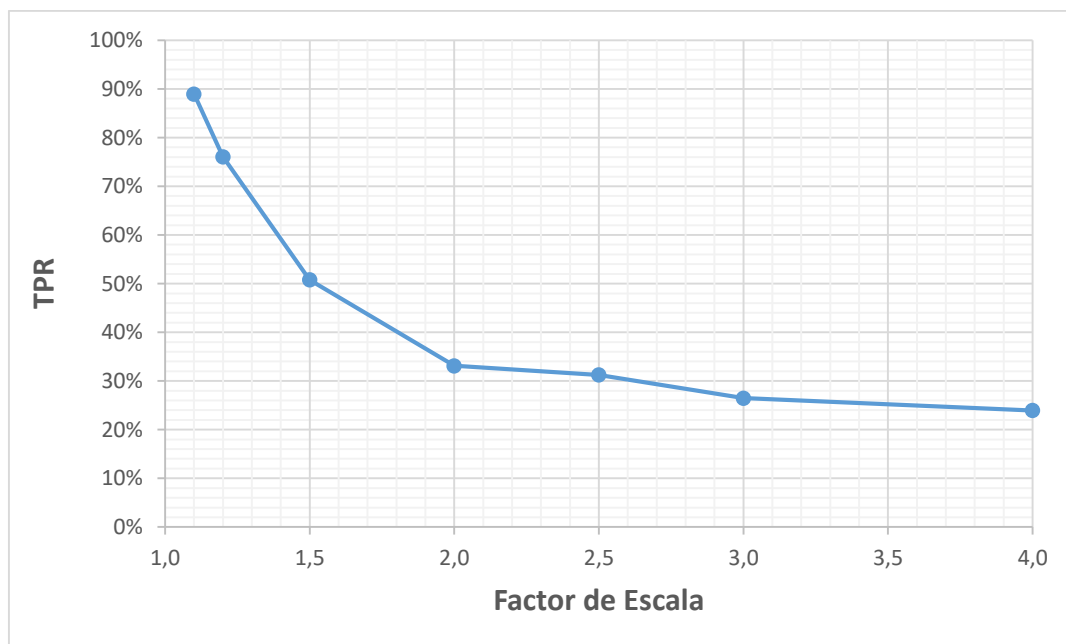


Figura 35. TPR respecto al factor de escala.

Tal y como hemos afirmado, si miramos las gráficas y la tabla comparativa, observamos que tanto el tiempo de ejecución como la precisión disminuyen conforme aumenta el factor. El tiempo de ejecución disminuye muy rápido aproximadamente entre 1.0 y 2.0, para luego presentar un decremento bastante lento. Algo parecido pasa con el TPR, aunque para factores altos podemos observar que el descenso en precisión es ligeramente superior que el descenso en tiempo de computación.

## 4.5 Pre-procesamiento y extracción de características

### 4.5.1 Escala de grises y color

En esta sección vamos a evaluar cómo afecta la representación de la imagen a los resultados y el clasificador. Por representación nos estamos refiriendo a si tratamos con imágenes a color con modelo RGB, o si solo utilizamos imágenes en escala de grises.

Dimensión	BASE	Grises u=0.3	Grises u=0.4	Grises u=0.5
<b>Resultados</b>				
<b>TP</b>	448	506	472	425
<b>FP</b>	321	522	387	280
<b>Tiempo medio de Test (ms)</b>				
<b>Imagen</b>	274.71	253.06	249.63	253.4
<b>I. Positiva</b>	426.05	392.42	387.86	392.63
<b>I. Negativa</b>	129.43	119.27	116.93	119.74
<b>Precisión</b>				
<b>FPPI</b>	0.5459	0.8878	0.6582	0.4762
<b>FPPW</b>	0.0010	0.0017	0.0012	0.0009
<b>TPR</b>	76.06%	85.91%	80.14%	72.16%
<b>FPR</b>	0.10%	0.17%	0.13%	0.09%

Tabla 3. Resultados de comparar color y escala de grises.



Figura 36. Curva ROC entorno a los valores del umbral de detección más relevantes.

Como podemos ver en la tabla, el tiempo medio de computación en la fase de ejecución es ligeramente inferior, aproximadamente 20 milisegundos menos. Sin embargo, la precisión general sobre el mismo umbral de detección es inferior, debido a que el FPR es superior (puntos negros en la curva ROC). Por esa razón hemos incluido en la comparación los resultados del umbral 0.4 y 0.5 (puntos blancos en la curva ROC). Podríamos obtener resultados muy parecidos a cuando utilizamos imágenes en color si usamos un valor para el umbral entre 0.4 y 0.5. Esto se puede apreciar mejor en la figura 36, que muestra la curva ROC en torno a los valores mencionados del umbral. La curva naranja representa el ratio TPR sobre FPR para la escala de grises, la curva azul representa al color.

#### 4.5.2 Corrección de color

En la configuración base que estamos utilizando, no se realiza ningún tipo de pre-procesamiento a las ventanas antes de extraer el Histograma de Gradientes. Ahora vamos a aplicar una corrección de color, también llamada corrección gama, antes de extraer el HOG. Extra corrección consiste en aplicar la operación de raíz cuadrada a los píxeles de cada canal RGB de la imagen por separado.

Pre-procesamiento	BASE	Corrección gamma
<b>Resultados</b>		
<b>TP</b>	448	450
<b>FP</b>	321	318
<b>Tiempo medio de Test (ms)</b>		
<b>Imagen</b>	274.71	285.16
<b>I. Positiva</b>	426.05	441.19
<b>I. Negativa</b>	129.43	135.36
<b>Precisión</b>		
<b>FPPI</b>	0.5459	0.5408
<b>FPPW</b>	0.0010	0.0010
<b>TPR</b>	76.06%	76.40%
<b>FPR</b>	0.10%	0.10%

Tabla 4. Resultados con y sin corrección de color.

Como podemos ver en la tabla comparativa de resultados, aplicar la corrección de color antes de extraer el HOG se traduce en un ligero incremento en la precisión (de 0.34% en el TPR), y un ligero incremento en el tiempo de computación requerido (aproximadamente 10ms), lo cual es lógico porque realizamos una tarea más.

### 4.5.3 Computación del HOG

Podemos conseguir comparaciones interesantes si variamos los parámetros de extracción del HOG. La librería OpenCV nos da mucha libertad a la hora de ajustar los parámetros. Entre otras cosas podemos elegir la cantidad de píxeles por celda, el número de celdas por bloque, la superposición de los bloques o el número de *bins* por histograma.

La configuración base utiliza bloques de 16x16 píxeles con 2x2 celdas por bloque. En esta sección vamos a utilizar también bloques de 8x8 y 32x32, con el mismo número de celdas, 2x2.

Hay que mencionar que cada vez que alteramos el tamaño del bloque, la dimensión final del vector de características también cambia. Por lo tanto, las variaciones obtenidas se diferencian principalmente en el tamaño del vector, o lo que es lo mismo, la cantidad de información que contienen las características extraídas.

Tamaño del bloque Tamaño del vector Nº de bloques	BASE 3780D 105	32x32 756D 21	8x8 16740D 465
<b>Tiempo de extracción de las características de entrenamiento</b>			
<b>Positivas</b>	17m 34s	4m 11s	1h 16m 15s
<b>Negativas</b>	20m 56s	5m 08s	1h 46m 58s
<b>Resultados</b>			
<b>TP</b>	448	444	494
<b>FP</b>	321	331	625
<b>Tiempo medio de Test (ms)</b>			
<b>Imagen</b>	274.71	280.21	781.7
<b>I. Positiva</b>	426.05	436.69	1220
<b>I. Negativa</b>	129.43	130.00	360.94
<b>Precisión</b>			
<b>FPPI</b>	0.5459	0.5629	1.0629
<b>FPPW</b>	0.0010	0.0011	0.0020
<b>TPR</b>	76.06%	75.38%	83.87%
<b>FPR</b>	0.10%	0.11%	0.20%

Tabla 5. Resultados comparativos del tamaño de los bloques.

De cada bloque sale siempre un vector 36D, porque siempre hay 4 celdas y 9 *bins*.

Al reducir el tamaño del bloque, aumentamos la cantidad de bloques que caben en la ventana de detección, lo que se traduce en un vector de características más grande. Como podemos ver en los resultados (8x8), el tiempo de computación incrementa significativamente (aproximadamente 500ms), y el incremento del TPR no se traduce en una mejora de precisión general, porque la tasa de falsos positivos se duplica, como se puede observar en los valores FPR, FPPW y FPPI.

Cuando se incrementa el tamaño del bloque, caben menos bloques en la ventana de detección, lo que se traduce en un vector de características más pequeño. Los resultados obtenidos para el bloque 32x32 son muy similares a los obtenidos con el bloque 16x16, pero ligeramente inferiores en todos los aspectos, TPR, FPR, y tiempo.

Los bloques de 16x16 ofrecen los mejores resultados para nuestro dataset (teniendo en cuenta solo tamaños fijos y cuadrados). Los bloques más pequeños o más grandes alcanzan peor rendimiento a la hora de capturar la visión general de una persona.

#### 4.5.4 Reemplazar HOG por LBP

La última variación que vamos a realizar en la extracción de características es la sustitución del Histograma de Gradientes por Local Binary Patterns (ver sección 2.2.5). Nuestros vectores de características LBP tienen un tamaño de 6195, debido a los parámetros escogidos. En la siguiente tabla podemos comprobar la gran diferencia que hemos encontrado entre HOG y LBP. La principal está en el tiempo de computación. El algoritmo en si para extraer las características es más lento, y teniendo en cuenta que se ejecuta 309850 veces (cantidad de imágenes del test), la diferencia es notable. El mayor tamaño del vector también contribuye a una computación más lenta. En cuanto a resultados, el altísimo FPR lastra el buen resultado del TPR. Como en ocasiones anteriores, lo ideal sería variar el umbral de detección para ver en qué punto obtenemos un FPPW (o TPR) parecido y comparar entonces el TPR. Por desgracia la falta de tiempo nos impide ejecutar tantas veces un test de varias horas. Sería interesante combinar HOG y LBP concatenando los dos vectores, pero tendríamos el mismo problema de tiempo que con LBP solo.

Tamaño del bloque Tamaño del vector	BASE 3780D	LBP 6195D
<b>Tiempo de extracción de las características de entrenamiento</b>		
<b>Positivas</b>	17m 34s	30m 39s
<b>Negativas</b>	20m 56s	38m 14s
<b>Resultados</b>		
<b>TP</b>	448	569
<b>FP</b>	321	1286
<b>Tiempo medio de Test (ms)</b>		
<b>Imagen</b>	274.71	27641.98
<b>I. Positiva</b>	426.05	43779.24
<b>I. Negativa</b>	129.43	12150.21
<b>Precisión</b>		
<b>FPPI</b>	0.5459	2.1871
<b>FPPW</b>	0.0010	0.0042
<b>TPR</b>	76.06%	96.60%
<b>FPR</b>	0.10%	0.42%

Tabla 6. Resultados comparativos entre HOG y LBP.



## 4.2 Clasificadores y sus parámetros

### 4.2.1 SVM: Parámetro C

El parámetro C le dice al optimizador de la SVM cuanto penalizar la clasificación incorrecta de cada ejemplo de entrenamiento. Para grandes valores de C, la optimización se inclinará por un hiperplano con menores márgenes, si ese hiperplano es más eficiente clasificando correctamente los ejemplos de entrenamiento. Un valor muy pequeño de C obligará al optimizador a buscar el hiperplano que separe los ejemplos con mayor margen, aunque clasifique más de forma incorrecta.

Hemos decidido evaluar diferentes grados de tolerancia a los fallos de clasificación cambiando el parámetro c. Con valores extremos de c, pequeño ( $c=2 \times 10^{-3}$ ) o alto ( $c=10$ ), el sistema no consigue converger. Con variaciones razonables obtenemos los resultados presentes en la siguiente tabla.

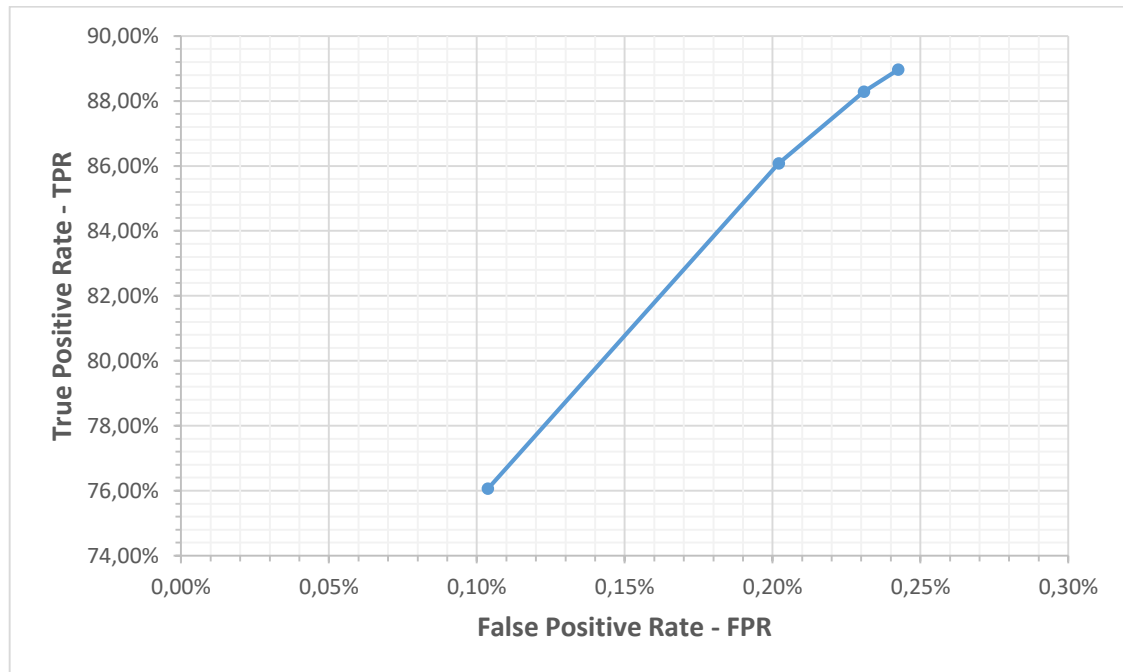
Valor de C	BASE	0.2	0.5	1.0
<b>Resultados de entrenamiento</b>				
<b>Score</b>	0.9861	1.0	1.0	1.0
<b>Tiempo (s)</b>	1.9375	3.1719	3.4375	3.5938
<b>Resultados</b>				
<b>TP</b>	448	507	520	524
<b>FP</b>	321	625	714	750
<b>Tiempo medio de Test (ms)</b>				
<b>Imagen</b>	274.71	272.35	272.8	272.56
<b>I. Positiva</b>	426.05	425.62	426.11	425.62
<b>I. Negativa</b>	129.43	125.21	125.62	125.62
<b>Precisión</b>				
<b>FPPI</b>	0.5459	1.0629	1.2143	1.2755
<b>FPPW</b>	0.0010	0.0020	0.0023	0.0024
<b>TPR</b>	76.06%	86.08%	88.29%	88.96%
<b>FPR</b>	0.10%	0.20%	0.23%	0.24%

Tabla 7. Resultados obtenidos de variar el parámetro C en una SVM Lineal.

El score hace referencia a la precisión del clasificador para clasificar los propios ejemplos de entrenamiento. El tiempo de entrenamiento excluye la carga de las características desde los ficheros (que en realidad siempre toma la mayor parte). Podemos ver que este último aumenta con la C y que para todos los valores excepto para la base el clasificador clasifica perfectamente sus propios ejemplos.

El tiempo de ejecución medio del test se puede considerar prácticamente el mismo, dado que las diferencias apenas alcanzan los dos milisegundos.

En cuanto a los resultados de precisión, el incremento del FPR vuelve a superar al de TPR, como hemos visto en anteriores comparaciones. El cambio no es tan notable entre los valores de 0.2, 0.5 y 1.0, pero si se compara cualquiera de ellos con el de referencia, la diferencia es mucho más evidente. Podemos apreciarlo mejor en la siguiente figura.



*Figura 37. Fragmento de la curva ROC para los valores de C examinados.*

Se ve reflejado en el score y en los resultados de test que a mayor valor de C más tiende el clasificador a sobreentrenar, dado que prioriza la clasificación correcta sobre el margen del hiperplano.

Si pudiéramos usar kernels no lineales, dispondríamos de nuevas combinaciones de parámetros que investigar, pero, al igual que la implementación que tomamos como referencia, usamos solo SVM lineales para beneficiarnos de su simplicidad y velocidad computacional.

### 4.2.2 Regresión Logística

El primer clasificador que vamos a comparar con la SVM es el de Regresión Logística. El tiempo de entrenamiento obtenido para este clasificador es de 5609 milisegundos, sin contar la carga de los vectores desde los ficheros. La precisión media sobre los ejemplos utilizados para entrenar es 0.99778.

Clasificador	BASE	RL $\mu=0.3$	RL $\mu=1.8$
<b>Resultados</b>			
<b>TP</b>	448	529	441
<b>FP</b>	321	752	312
<b>Tiempo medio de Test (ms)</b>			
<b>Imagen</b>	274.71	271.92	308.83
<b>I. Positiva</b>	426.05	424.15	479.00
<b>I. Negativa</b>	129.43	125.78	145.47
<b>Precisión</b>			
<b>FPPI</b>	0.5459	1.2789	0.5306
<b>FPPW</b>	0.0010	0.0024	0.0010
<b>TPR</b>	76.06%	89.81%	74.87%
<b>FPR</b>	0.10%	0.24%	0.10%

Tabla 8. Resultados de la comparación con Regresión Logística.

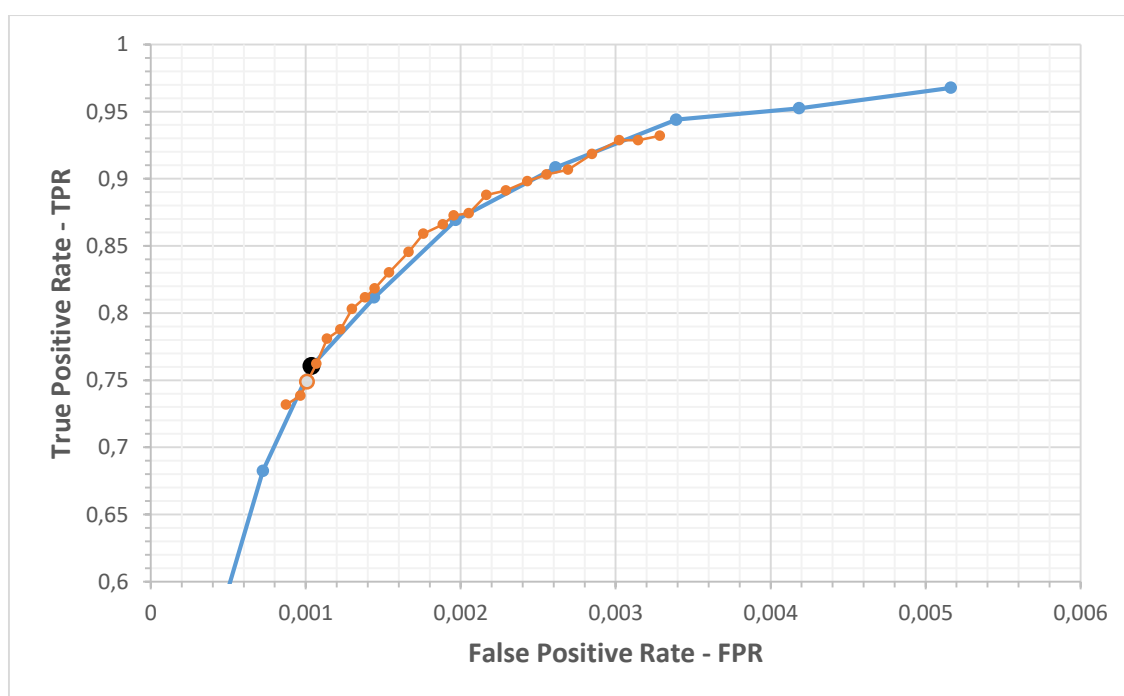


Figura 38. Curva ROC de la SVM y la Regresión Logística para distintos valores del umbral.

Como antes al comparar los resultados obtenidos con imágenes a color y con escala de grises, tenemos que cambiar el umbral del clasificador nuevo para obtener unos resultados parecidos a los de referencia. Como antes, podemos observar en detalle con una curva ROC como varía el ratio TPR sobre FPR según el umbral para ambos modelos en la figura 38. En la gráfica, el punto verde sobre la línea naranja equivale al valor de umbral 1.8 en el clasificador de regresión, y el punto negro sobre la línea azul equivale al umbral de referencia (0.3) en el clasificador de referencia (SVM). Merece la pena destacar que el ratio varía mucho menos en la Regresión Logística cuando se cambia el umbral.

Estudiando los demás resultados podemos comprobar que para un valor de precisión casi idéntico tanto en TPR (76.06% en SVM frente a 74.87% en RL) como en FPR (0.10% para ambos) el tiempo de computación es sensiblemente superior (aproximadamente 34 milisegundos), por lo que podemos considerar el modelo SVM antes que el de Regresión a la hora de formar un detector para nuestro dataset.

### 4.2.3 Perceptrón Multicapa

Con un simple vistazo a los resultados de la siguiente tabla ya nos damos cuenta de que el Perceptrón Multicapa es muy diferente a los dos clasificadores previos, y no funciona muy bien con el Histograma de Gradientes Orientados como características.

Clasificador	BASE	MLP	MLP	MLP	MLP
Activation	-	relu	relu	logistic	logistic
Solver	-	adam	lbfgs	adam	lbfgs
Resultados de entrenamiento					
Score	0.9861	0.4471	1.0	0.5531	0.5531
Tiempo (s)	1.9375	1426.08	458.61	127.66	15.09
Resultados					
TP	448	586	553	0	0
FP	321	9236	1000	0	0
Tiempo medio de Test (ms)					
Imagen	274.71	2588.25	2320.37	2432.72	2376.12
I. Positiva	426.05	3916.12	3590.01	3732.75	3711.91
I. Negativa	129.43	1313.49	1101.51	1184.69	1093.75
Precisión					
FPPI	0.5459	15.7075	1.7007	0.0000	0.0000
FPPW	0.0010	0.0298	0.0032	0.0000	0.0000
TPR	76.06%	99.49%	93.89%	0.00%	0.00%
FPR	0.10%	2.99%	0.32%	0.00%	0.00%

Tabla 9. Resultados de la comparación con Perceptrón Multicapa.

Desafortunadamente, no hemos podido probar una gran variedad de combinaciones con los parámetros por falta de tiempo. Para este test contamos con una única configuración de neuronas, con dos funciones de activación, y con dos resolvedores para optimizar los pesos.

Empezando con las funciones de activación, hemos probado la logística (función sigmoide) y la relu (rectified linear unit function). Salta a la vista los malísimos resultados obtenidos con la activación logística. Literalmente no se llega a detectar nada en la fase de test. En el caso de la activación relu, los resultados dependen ampliamente del resolvedor elegido. El *lbfgs* es un optimizador de la familia de los métodos quasi-Newtonianos. El *adam* es un optimizador basado en el gradiente estocástico. Según la documentación de sklearn, el *lbfgs* converge antes y obtiene una mejor precisión para datasets más pequeños, y el Adam se comporta mejor con datasets grandes (de miles de ejemplos o más). Nuestro dataset no es excesivamente grande y podemos confirmar con los resultados la afirmación anterior. Aun así, el FPR es demasiado alto (0.32%) y no compensa el 93.89% del TPR. Quizás bajando el umbral podríamos obtener resultados que se puedan tener en cuenta.

En los resultados de entrenamiento podemos comprobar que este clasificador ha sido el primero en necesitar más de unos pocos segundos para entrenar. Casi 24 minutos para la combinación *relu + adam*. Además, la precisión para clasificar los ejemplos de entrenamiento apenas supera el 50% o ni siquiera llega, salvo por la excepción de la combinación que mejor puntuación ha conseguido luego en el test.

Dejando de un lado la precisión, el principal problema de éste clasificador con este vector de características y este dataset es el tiempo de computación. Una media de más de dos segundos y medio por imagen hace que sea muy difícil considerar su utilización más adelante en una aplicación más elaborada y realista.

#### 4.2.4 KNN. Vecinos más cercanos.

El problema que hemos visto en la sección anterior con el Perceptrón Multicapa se acentúa aún más ahora con el algoritmo de clasificación KNN. Los tiempos de ejecución son tan altos que no podemos ofrecer el mismo nivel de resultados que con los clasificadores anteriores por falta de tiempo para su ejecución, y porque profundizar en un algoritmo que necesita tanto tiempo de computación no tiene mucho sentido. A pesar de haber entrenado 8 modelos distintos ( $k = 1, 3, 5, 7$  y distancia de Manhattan o Euclídea), solo hemos probado los clasificadores  $k = 1$  con un dataset mucho más pequeño de apenas 35 imágenes. Podemos ver los resultados en las siguientes tablas.

Clasificador	BASE	KNN	KNN
K	-	1.0	1.0
Distancia	-	Manhattan	Euclídea
<b>Cantidad de imágenes y ventanas</b>			
N	588	35	35
W	309850	8480	8480
<b>Resultados</b>			
TP	448	36	25
FP	321	31	16
<b>Tiempo medio de Test (ms)</b>			
Imagen	274.71	12902.23	14054.91
I. Positiva	426.05	29845.05	32540.36
I. Negativa	129.43	4062.50	4410.33
<b>Precisión</b>			
FPPI	0.5459	0.8857	0.4571
FPPW	0.0010	0.0037	0.0019
TPR	76.06%	90.00%	62.50%
FPR	0.10%	0.37%	0.19%

Tabla 10. Resultados de test para KNN con  $k = 1$ .

K	1.0	1.0	3.0	3.0
Distancia	Manhattan	Euclídea	Manhattan	Euclídea
<b>Resultados de entrenamiento</b>				
Score	1.0	1.0	0.9362	0.8228
Tiempo (s)	2.1094	2.2969	2.1094	2.0938

Tabla 11. Resultados de entrenamiento para KNN con  $k = 1$  y  $k = 3$ .

K	5.0	5.0	7.0	7.0
Distancia	Manhattan	Euclídea	Manhattan	Euclídea
<b>Resultados de entrenamiento</b>				
Score	0.9173	0.7854	0.9099	0.7714
Tiempo (s)	2.0938	2.0781	2.0625	2.0625

Tabla 12. Resultados de entrenamiento para KNN con  $k = 5$  y  $k = 7$ .

Hay que comentar que el tiempo medio especificado en los resultados de test “apenas” es de 13, 14 segundos porque las 35 imágenes utilizadas son de las más pequeñas del dataset original. Las imágenes más grandes suele tardar más de un minuto en clasificarlas. Está claro que el vector HOG es demasiado grande para computar de forma eficiente el algoritmo KNN. Esto en absoluto quiere decir que KNN no sirva para clasificar imágenes. Lo que hay que hacer es buscar un vector de características más eficiente (como por ejemplo histogramas de color) o incluso utilizar algún algoritmo de reducción de características. Lo mismo podemos decir del MLP.

## 5. Aplicación Práctica

El objetivo de esta sección es crear un detector funcional para vídeos (no excesivamente complicados) sirviéndonos de las observaciones, resultados y estudios realizados a lo largo del trabajo. Queremos saber si con lo que hemos visto hasta ahora se puede conseguir un detector mínimamente funcional.

Primero vamos a hacer uso únicamente de uno de los detectores entrenados, ajustando sus parámetros. Después vamos a mejorarlo combinando el detector con otras técnicas de visión artificial, como el tracking.

### 5.1 Solo el detector

El algoritmo es simple. Resumidamente, tenemos que leer un fichero de video, y por cada frame, escanearlo y dibujarlo con los rectángulos pertenecientes a las personas detectadas. Para el detector sólo necesitamos las funciones *testImage()*, *pyramid()*, *sliding\_window()*, *non\_max\_suppression()* y *extractHOGfeatures()*. Las cuatro últimas se utilizan en la primera, que es la única que vamos a utilizar desde el bucle principal. Como ya hemos visto antes, *testImage()* escanea una imagen y devuelve las cajas que contienen a las personas de la imagen.

Como características vamos a utilizar el Histograma de Gradientes Orientados (con bloques de 16x16 y con corrección gamma) y como clasificador vamos a utilizar la Máquina de Soporte de Vectores (con  $c = 0.01$ ). El umbral de detección y el factor de escala lo vamos a ajustar alrededor del 0.3 y el 1.1 respectivamente. Sobre el frame vamos a realizar una serie de operaciones aunque al final siempre vamos a dibujar en pantalla el original (más el texto o los rectángulos que correspondan). Por ejemplo, al detector le vamos a pasar el frame reducido a escala de grises y además si la anchura del frame de video supera los 360 píxeles lo reescalaremos a esta dimensión (la altura se hace de forma proporcional). Este paso es importante porque como ya hemos visto, el tamaño de la imagen afecta en gran medida al rendimiento del detector.

Para medir el rendimiento computacional vamos a utilizar una métrica llamada frames por segundo (FPS). Esto indica cuantos frames se leen, procesan y muestran durante un periodo de un segundo. En la siguiente figura podemos ver un esquema general del algoritmo principal.

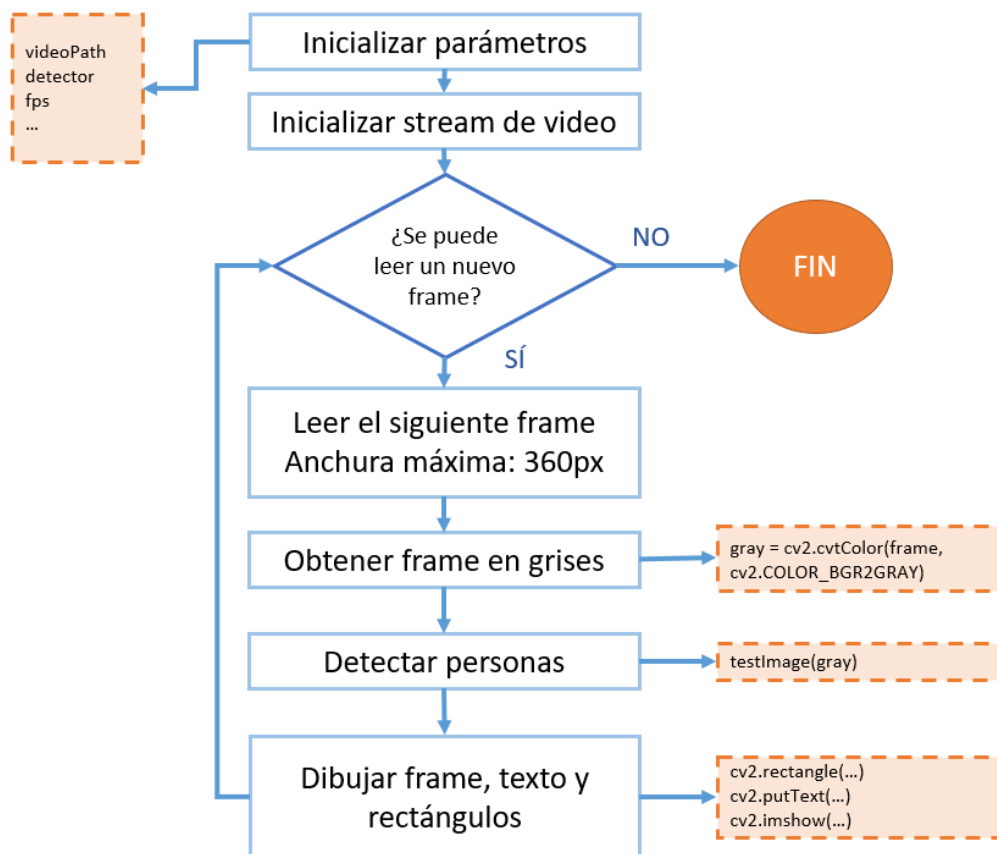


Figura 39. Esquema general del detector.

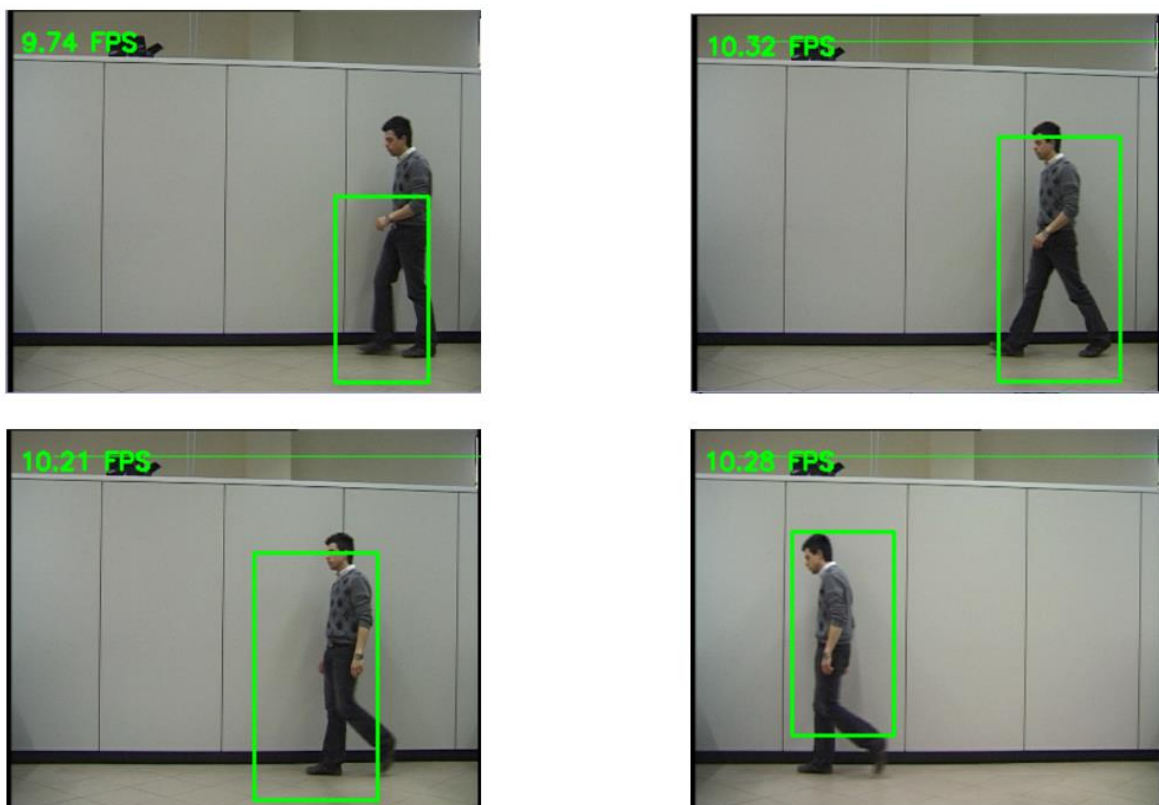


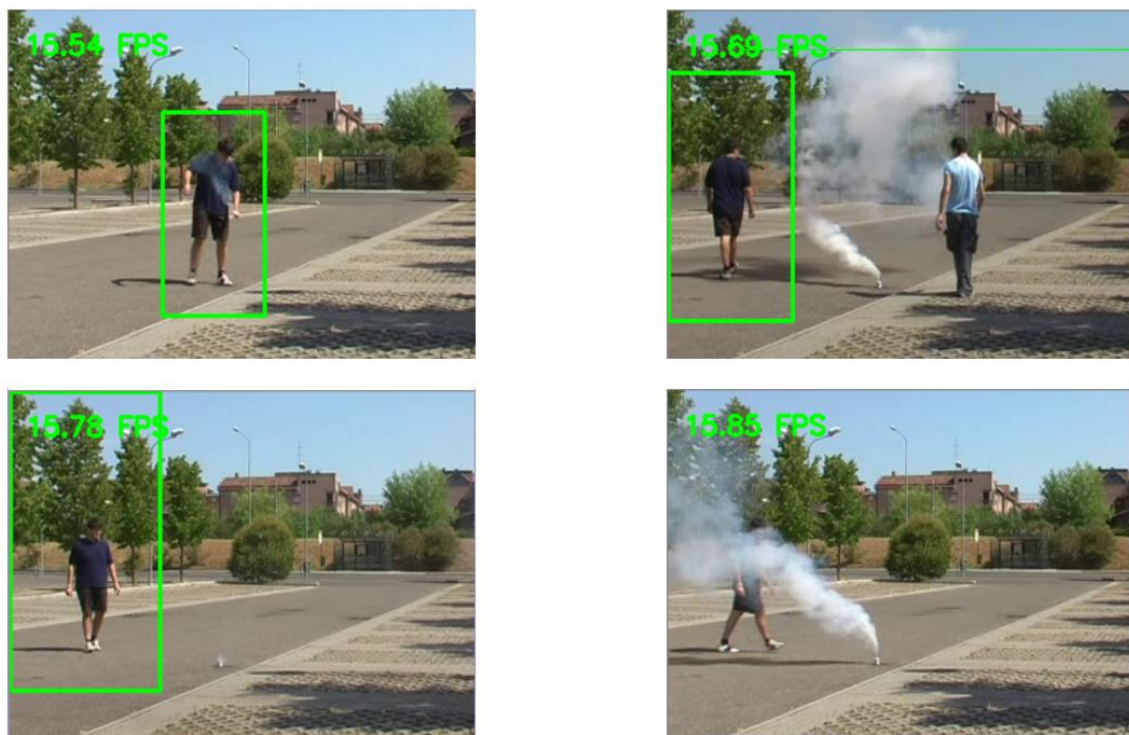
Figura 40. Ejecución del detector sobre un video sencillo.



En la figura 40 podemos ver cuatro capturas de un ejemplo de ejecución del detector. El video que hemos utilizado intencionadamente para empezar no ofrece un gran nivel de dificultad al detector. Se trata de un fondo bastante plano y una persona pasa simplemente de derecha a izquierda caminando.

Si analizamos el rendimiento vemos que la reproducción alcanza aproximadamente 10 FPS. Teniendo en cuenta que el video original tiene 25 FPS, es como si lo viéramos a cámara lenta. La computación requerida por la detección lastra la reproducción normal del video.

Si por otra parte, analizamos la precisión, podemos darnos cuenta de que aunque es capaz de detectar a la persona durante todo su recorrido el detector no produce rectángulos de tamaño y posición consistentes, lo que se traduce en marcas diferentes para frames consecutivos, como podemos apreciar en las dos imágenes superiores de la figura 40.



*Figura 41. Ejecución del detector sobre video más elaborado.*

Si ejecutamos el detector sobre un vídeo más elaborado, rápidamente nos damos cuenta de que la precisión dista de ser perfecta. En la figura anterior podemos ver como el detector puede dejarse personas, muestra rectángulos muy distintos en tamaño para la misma persona, y también falla cuando la persona es parcialmente ocluida en la imagen.

## 5.2 Detector + tracking

Se llama **tracking** a la acción de localizar un objeto en frames sucesivos [11]. No vamos a entrar en detalle en los distintos algoritmos de tracking ni en la base teórica, dado que no es el objetivo del trabajo. Simplemente nos vamos a limitar a explicar que beneficios aporta, por qué, y cómo utilizarlo para alcanzar nuestro objetivo con el detector. Estas son algunas de las razones por las que a veces utilizar un algoritmo de tracking puede ser mejor que utilizar uno de detección.

- El tracking es más rápido que la detección: Habitualmente los algoritmos de tracking son más rápidos que los de detección. La razón es simple. Cuando estás siguiendo un objeto que ya has detectado en el frame anterior, sabes muchas cosas sobre su apariencia, posición, dirección y velocidad. En el siguiente frame puedes utilizar toda esa información para predecir la nueva posición del objeto y realizar una búsqueda rápida por los alrededores. Un algoritmo de tracking utiliza toda la información que ya tiene sobre el objeto para localizarlo, mientras que un algoritmo de detección comienza de cero en cada frame. Por lo tanto, para diseñar un sistema eficiente lo habitual es ejecutar el detector cada  $n$  frames y utilizar tracking en el resto. ¿Por qué no utilizar entonces detección al principio y tracking en el resto del vídeo? Porque los algoritmos de tracking pueden perder el objeto si desaparece de la imagen, o si es ocluido, y además suele acumular error en cada frame. El detector ayuda a solventar estos problemas ejecutándose de vez en cuando.
- El tracking puede ayudar cuando la detección falla: Si utilizamos un detector de personas en un video y la persona se oculta parcialmente con otro objeto, el detector con mucha probabilidad fallará. Un buen algoritmo de tracking es capaz de lidiar con cierto grado de oclusión.
- El tracking preserva la identidad: La salida de un detector de objetos es un array de rectángulos que contienen a los objetos. Sin embargo, no hay identidad asociada al objeto. Por ejemplo, si ejecutamos el detector sobre un conjunto de diez pelotas, en el primer frame la pelota uno puede estar representada por el rectángulo uno, y en el segundo frame por el rectángulo 6. Al usar detección no sabemos que rectángulo corresponde con qué objeto en dos frames consecutivos. Por otro lado, con el tracking el rectángulo que corresponde con la pelota uno en el primer frame corresponderá con la misma pelota en el frame  $N$ .

La librería OpenCV ofrece hasta seis tipos de algoritmos de tracking diferentes: Boosting, MIL, KCF, TLD, MEDIAFLOW y GOTURN. Hemos decidido utilizar KCF porque ofrece un buen equilibrio entre rendimiento y precisión. En la siguiente figura podemos ver cómo cambia la estructura general del detector.

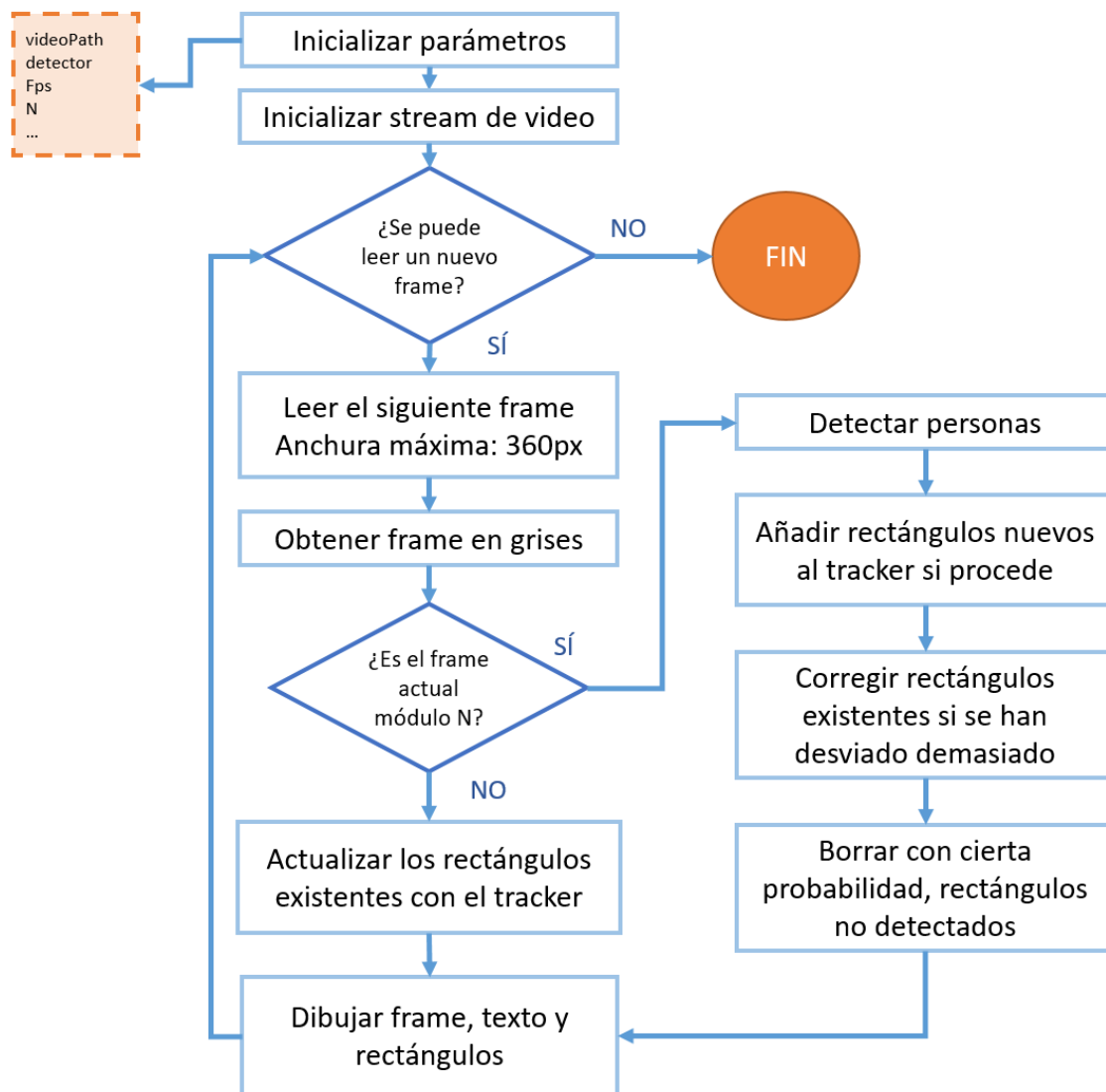
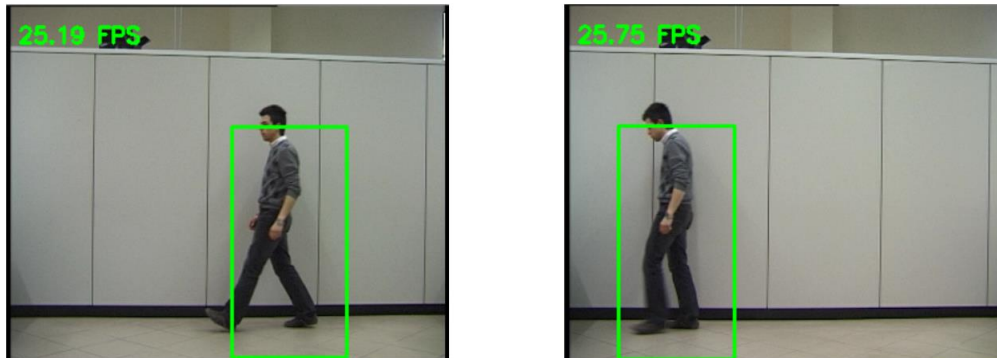


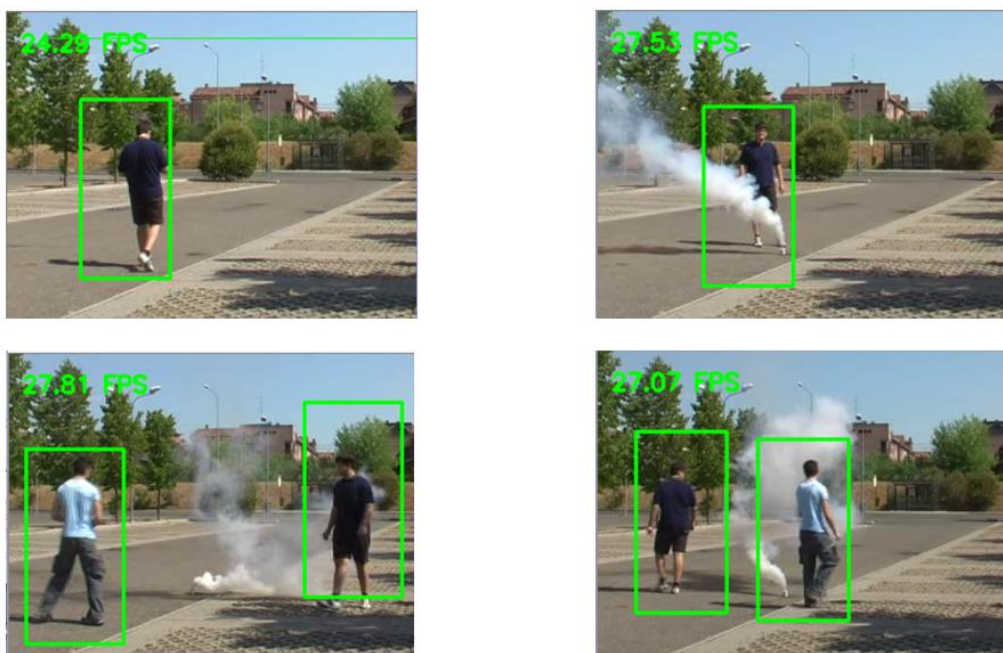
Figura 42. Estructura general del detector con tracking.

Cada N frames (nosotros utilizamos 12), detectamos las personas, actualizamos los rectángulos existentes si se corresponden con esas personas y se han desviado demasiado, añadimos rectángulos por las personas nuevas detectadas, y borramos con cierta probabilidad los rectángulos que no se han podido asociar a ninguna detección. El resto de frames simplemente actualizamos la posición de los rectángulos que tenemos con el tracker. La razón por la que utilizamos una distribución de probabilidad para borrar los rectángulos es la de mitigar los fallos en la detección. Las probabilidades utilizadas son muy simples. Si el rectángulo está en los bordes de la

imagen tiene un 80% de probabilidades de ser borrado. En caso contrario, la probabilidad es del 25%. Con más tiempo podríamos desarrollar un mejor sistema de probabilidades para el caso en el que el rectángulo está lejos de los bordes. Por ejemplo, podríamos cuantificar cuanto tiempo tiene el rectángulo para dar más probabilidades de permanecer a los que más tiempo tienen y viceversa. De esta forma se penalizaría la permanencia de los falsos positivos que se dan menos a menudo y se premiarían las detecciones correctas.



*Figura 43. Ejecución del detector sobre un video sencillo.*



*Figura 44. Ejecución del detector sobre un video más complejo.*

Como podemos ver en las figuras, el detector ha mejorado pero no es perfecto. Los rectángulos son más consistentes entre frames, sigue habiendo fallos de detección aunque menos (oclusión, dos personas muy juntas, etc.) y sigue habiendo problemas de falsos positivos (aquí por ejemplo el humo). En cuanto al rendimiento, los FPS han mejorado notablemente y llegan hasta el límite máximo del vídeo.

## 6. Conclusiones

### 6.1 Conclusiones técnicas

El objetivo de este proyecto era el de realizar un estudio del comportamiento de varias técnicas de clasificación y características a la hora de utilizarse para detectar personas.

En la sección de resultados ya se han realizado conclusiones particulares de las configuraciones, clasificadores y características evaluadas. En esta sección resumimos la conclusión general que extraemos del trabajo realizado.

Respecto a las características el Histograma de gradientes tiene la capacidad de “capturar” la forma general de una persona independientemente de su ropa, de la iluminación, o de los colores en general, y ofrece muchos parámetros que permiten ajustarlo para conseguir mayor y menor precisión y eficiencia. Sin embargo, sufre bastante cuando la persona es parcialmente ocluida o adopta una pose que se sale de lo común (agacharse por ejemplo, aunque también depende del dataset utilizado). El LBP no es el mejor método para detectar personas, lo que no quiere decir que no se pueda usar para otro tipo de detecciones. Es necesario obtener un vector de características más pequeño, y que las diferencias del dataset estén principalmente basadas en las texturas. Puede servir por ejemplo para diferenciar suelos de baldosas, papel de regalo y teclados. Todos esos objetos tienen patrones fácilmente reconocibles y bien diferenciados.

Respecto a los clasificadores las Máquinas de Soporte de Vectores obtienen los mejores resultados (que distan de ser perfectos) y se ha demostrado que la Regresión Logística también se puede utilizar para este cometido. El perceptrón multicapa y el clasificador KNN han alcanzado resultados bastante pobres. Como con el LBP, esto no significa que no se pueden utilizar para tareas de clasificación. Hemos visto que el MLP depende mucho de sus parámetros (función de activación, optimizador de pesos, configuración de neuronas, etc.). Las redes neuronales pueden funcionar muy bien por ejemplo para la detección de dígitos con los valores de los píxeles directamente como características (después de aplicar umbralización). El KNN podríamos utilizarlo en este tipo de tareas o parecidas si lo combináramos con otro vector de características, por ejemplo el histograma de color.

Respecto a la utilización del detector en un entorno real, hemos podido comprobar que por sí solo tiene deficiencias tanto en el rendimiento como en la precisión. Se puede llegar a conseguir un detector mínimamente funcional si se combina la detección con otros métodos como el tracking o la detección de movimiento (más útil en entornos donde la imagen de fondo es más estática, como la de una cámara de seguridad).

En resumen, la detección de personas es un campo muy extenso y útil, con muchas opciones que explorar.

## 6.2 Trabajo futuro

Centrándonos en el presente trabajo, hay varias posibles direcciones que podemos tomar para conseguir mejoras.

Primero, agrandar el dataset de test y de entrenamiento, tanto en complejidad como en dimensión, podría ser una práctica interesante. Merece la pena entrenar y probar el detector con datasets más grandes, con más ejemplos positivos y negativos. El dataset de personas INRIA contiene un número limitado de ejemplos positivos por lo que solo sería posible incrementar el número de ejemplos negativos. Sin embargo, también podríamos probar otros datasets de personas. Podemos encontrar datasets más simples, con fondos planos y un rango de poses más limitado, y también datasets más complejos, con una amplia gama de poses o incluso oclusiones.

Luego, por supuesto, podríamos realizar muchas más pruebas de configuración con los clasificadores utilizados. Como ya hemos dicho, por ejemplo podríamos probar más configuraciones de neuronas con el MLP, histogramas de color u otras características con KNN, además de distintos valores para la  $k$  y más métricas para las distancias.

Otra opción es combinar lo ya utilizado con otras técnicas de aprendizaje. Por ejemplo podríamos implementar un detector con una cascada de HOGs [12]. Muy resumidamente esta técnica se basa entre otras cosas en la idea de que los HOGs de tamaño fijo no son capaces de capturar con precisión un objeto complejo tanto a pequeña como a gran escala. Tiene en cuenta muchos más bloques de tamaños variables y proporciones 1:1, 1:2 y 2:1. Pero no los utiliza todos. Construye un clasificador fuerte de varios niveles con los mejores clasificadores débiles (los SVM y bloques que mejor resultados obtienen) mediante Boosting.

Por último, como ya hemos comentado antes, podemos explorar la idea de construir un detector a tiempo real combinando este con otras técnicas de pre-procesado y post-procesado. Mejorar el tracking, detectar el movimiento, separar el fondo, detectar clusters, o en general utilizar la información espacial y temporal son solo algunas de las opciones.



## 7. Anexo

### 7.1 Lista de figuras

Figura 1. Esquema general de un sistema de detección de imágenes. ....	4
Figura 2. Imagen original, gradiente horizontal $\partial x(x_i, y_i)$ y gradiente vertical $\partial y(x_i, y_i)$ . ....	9
Figura 3. Magnitud (m) y dirección (Theta) del gradiente. ....	9
Figura 4. Ejemplo de cálculo del histograma para una celda. ....	10
Figura 5. Histograma de gradientes correspondiente al ejemplo de la figura 4. ....	11
Figura 6. Imagen original y su correspondiente Histograma de Gradientes. ....	11
Figura 7. Cálculo de LBP. ....	12
Figura 8. Tratamiento de los píxeles de los bordes y esquinas. ....	13
Figura 9. Tres ejemplos de vecindad con p y r variables. ....	13
Figura 10. Tarea de asociar una instancia de entrada x, con un vector de características v, a una clase y. ....	16
Figura 11. Representación de una SVM de dos dimensiones. Elegimos el plano h1, que maximiza el margen. ....	19
Figura 12. Ejemplo de recta de regresión sobre un conjunto de datos de una única variable. ....	21
Figura 13. Ejemplo visual de KNN con dos variables y dos clases. ....	23
Figura 14. Estructura jerárquica de un sistema basado en ANS. ....	25
Figura 15. Estructura de una neurona artificial. ....	26
Figura 16. Tabla con las principales funciones de activación. ....	28
Figura 17. Estructura del perceptrón multicapa. ....	29
Figura 18. Ejemplo de imágenes positivas y negativas del dataset INRIA. ....	31
Figura 19. Representación de la generación de ejemplos positivos (Las dimensiones reales de las imágenes son aproximadas). ....	31
Figura 20. Representación de generación de ejemplos negativos. ....	31
Figura 21. Estructura general del detector. ....	32
Figura 22. Movimiento del bloque a través de la ventana de detección. ....	33
Figura 23. Ejemplo de una pirámide de imágenes. En cada capa de la pirámide la imagen se reduce y (opcionalmente) se suaviza. ....	35
Figura 24. Ejemplo de detección mediante pirámide y ventana deslizante. Se han omitido algunas capas. Este ejemplo produce una detección en la capa 4 y otra en la 2. Ambas sobre la persona de la imagen. ....	35

Figura 25. Resultado de aplicar o no NMS a las detecciones positivas al final del proceso. A la derecha vemos que en dos escalas diferentes se ha detectado a la misma persona, lo que produce dos positivos cuando solo debería ser uno.....	36
Figura 26. Especificación del ordenador utilizado para ejecutar el algoritmo.....	37
Figura 27. Esquema de ejecución de la función trainModel() .....	40
Figura 28. Esquema de ejecución de la función testImage() .....	42
Figura 29. Parámetros del clasificador de referencia o base. ....	44
Figura 30. Curva ROC. En rojo aparece el punto correspondiente al umbral base 0.3.....	46
Figura 31. Valor de TP y FP según el umbral escogido. ....	47
Figura 32. Cantidad de imágenes escaneadas con éxito según el Umbral. El punto más alto se corresponde con un umbral de 0.5, 374 imágenes escaneadas exitosamente. Un 63.61%.....	47
Figura 33. Escalado de la imagen. En este ejemplo un factor de escala 2 aumentaría las posibilidades de hacer coincidir la ventana deslizante con la persona. ....	48
Figura 34. Tiempo de ejecución respecto al factor de escala. ....	49
Figura 35. TPR respecto al factor de escala.....	49
Figura 36. Curva ROC entorno a los valores del umbral de detección más relevantes. ....	50
Figura 37. Fragmento de la curva ROC para los valores de C examinados. ....	55
Figura 38. Curva ROC de la SVM y la Regresión Logística para distintos valores del umbral. ....	56
Figura 39. Esquema general del detector. ....	61
Figura 40. Ejecución del detector sobre un video sencillo.....	61
Figura 41. Ejecución del detector sobre video más elaborado.....	62
Figura 42. Estructura general del detector con tracking.....	64
Figura 43. Ejecución del detector sobre un video sencillo.....	65
Figura 44. Ejecución del detector sobre un video más complejo. ....	65



## 7.2 Lista de tablas

Tabla 1. Resultados del clasificador base.....	44
Tabla 2. Resultados de los distintos factores de escala. El factor base es 1.2. ....	48
Tabla 3. Resultados de comparar color y escala de grises. ....	50
Tabla 4. Resultados con y sin corrección de color. ....	51
Tabla 5. Resultados comparativos del tamaño de los bloques. ....	52
Tabla 6. Resultados comparativos entre HOG y LBP.....	53
Tabla 7. Resultados obtenidos de variar el parámetro C en una SVM Lineal. ....	54
Tabla 8. Resultados de la comparación con Regresión Logística. ....	56
Tabla 9. Resultados de la comparación con Perceptrón Multicapa.....	57
Tabla 10. Resultados de test para KNN con $k = 1$ .....	59
Tabla 11. Resultados de entrenamiento para KNN con $k = 1$ y $k = 3$ . ....	59
Tabla 12. Resultados de entrenamiento para KNN con $k = 5$ y $k = 7$ . ....	59

## 8. Referencias

- [1] R. Szeliski, Object detection in Computer Vision: Algorithms and Applications, pp. 658-666.
- [2] M. Pechenizkiy, S. Puuronen, A. Tsymbal, FEATURE EXTRACTION FOR CLASSIFICATION IN THE DATA MINING PROCESS, p. 271.
- [3] N. Markuš, M. Frljak, P. S. Igor, J. Ahlberg y R. Forchheimer, A method for object detection based on pixel intensity comparisons organized in decision trees.
- [4] G. N. Srinivasan, and Shobha G. Statistical Texture Analysis, 2008.
- [5] Satya Mallick, Histogram of Oriented Gradients, 2016.
- [6] César Troya Sherdek, LBP y ULBP – Local Binary Patterns y Uniform Local Binary Patterns
- [7] Taiwo Oladipupo Ayodele, Types of Machine Learning Algorithms, University of PortsMouth, United Kingdom. Ch. 3, p. 19.
- [8] D. Boswell, Introduction to support vector machines, 6 August 2002.
- [9] Abdelmalik Moujahid, Iñaki Inza y Pedro Larrañaga, Tema 5. Clasificadores K-NN, Universidad del País Vasco.
- [10] Navneet Dalal and Bill Trigs, Histogram of Oriented Gradients for Human Detection.
- [11] Satya Mallick, Object Tracking with OpenCV and Python, 2017.
- [12] Qiang Zhu, Shai Avidan, Mei-Chen Yeh, and Kwang-Ting Cheng. Fast Human Detection Using a Cascade of Histograms of Oriented Gradients, 2006.